Time-independent geometrical deformation for elastic contacts

Camille Brunel

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ECOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

SPÉCIALITÉ : INFORMATIQUE

Par **Camille BRUNEL**

Déformations géométriques indépendantes du temps pour les contacts élastiques

Sous la direction de : **Pascal BARLA**
Co-encadrants : **Pierre BÉNARD, Gaël GUENNEBAUD**

Soutenue le 12 octobre 2021

Membres du jury :

M. Loïc BARTHE	Professeur	Université de Toulouse	Président du jury
Mme. Maud MARCHAL	Professeure	Université de Rennes	Rapporteure
M. Damien ROHMER	Professeur	Ecole Polytechnique	Rapporteur
Mme. Stefanie HAHMANN	Professeure	Université de Grenoble INP	Examinatrice
M. Pascal BARLA	Chargé de Recherche	Inria Université de Bordeaux	Directeur de thèse
M. Pierre BÉNARD	Maître de Conférence	Université de Bordeaux	Co-encadrant de thèse
M. Gaël GUENNEBAUD	Chargé de Recherche	Inria Université de Bordeaux	Co-encadrant de thèse

Thèse réalisée au Centre de Recherche Inria Bordeaux - Sud-Ouest,
au sein de l'équipe projet MANAO.

Thèse réalisée au Laboratoire Bordelais de Recherche en Informatique,
au sein du département Image et Son.

UMR 5800 Université, 33000 Bordeaux, France.

Abstract

As animated films and series become more and more present in the mainstream entertainment, the artists' needs are growing in term of fast and intuitive animation tools. Artists not only heavily rely on their imagination and skills to bring digital models to life; they also take inspiration from the physical world to better immerse viewers in their virtual environment.

Many objects of our everyday surroundings exhibit elastic deformations when put in contact with others, e.g., a stress ball crushed by a hand, a pillow smashing a head during a pillow fight or a soft ball bouncing on a goal post. They most notably tend to squash inside the contact and to bulge outside of it. Such squashing and bulging effects are essential to communicate plausible deformation while capturing the physical behavior of soft materials in a variety of contexts, such as animated films. This type of deformation is, however, notoriously difficult and tedious to manually reproduce by computer graphics (CG) artists, and existing tools remain limited for artistic use.

In practice, such deformations are thus generated through physically based simulation methods. However, due to their time-dependency, physical simulations must be run after the rigging and animation steps, preventing non-linear editing of the 3D scene. Moreover, artists also often resort to cartoonish deformation effects to better convey emotions and thoughts. Such exaggerated effects are difficult to achieve through physical simulations.

The main contribution of this thesis is a novel purely geometric deformation framework that assists the artist by resolving local contacts between elastic objects and producing bulge effects in an art-directable way. To achieve a seamless integration within animation workflows, we designed our deformation tool to provide instant feedback to the artist while enabling non-linear editing thanks to a fully time-independent strategy. To produce plausible bulge effects, our method can also preserve the volume exactly, while artistic controls are also possible to explore more exaggerated behaviors. More specifically, starting from multiple meshes in intersection, our deformer first computes the parts of the surfaces remaining in contact, and then applies a procedural displacement controlled by a profile curve. Although our tool processes each frame independently, it achieves temporally continuous deformations with artistic control of the bulge through a small number of pseudo-stiffness parameters. The plausibility of the deformation is further enhanced by anisotropically spreading the volume-preserving bulge. An extension is also proposed to handle self-collisions between adjacent parts of the same object that often occur in character skinning animation.

The result of this work is a robust, real-time deformer that can handle complex geometric configurations like a ball squashed by a hand, colliding lips, bending fingers, etc.

Keywords: 3D animation, mesh deformation, collision response, computer graphics, geometry processing.

Résumé

Les films et séries d'animation étant de plus en plus présents dans le divertissement grand public, les besoins des artistes en terme d'outils d'animation rapides et intuitifs ne cessent de croître. Les artistes ne font pas seulement appel à leur imagination et à leurs compétences pour donner vie à aux modèles numériques, ils s'inspirent également du monde physique pour mieux immerger les spectateurs dans leur environnement virtuel.

De nombreux objets de notre environnement quotidien présentent des déformations élastiques lorsqu'ils sont mis en contact avec d'autres, par exemple une balle anti-stress écrasée par une main, un oreiller écrasant une tête lors d'une bataille d'oreillers ou un ballon rebondissant sur un poteau de but. Ils ont notamment tendance à s'écraser à l'intérieur du contact et à gonfler à l'extérieur. Ces effets d'écrasement et de gonflement sont essentiels pour communiquer une déformation plausible tout en capturant le comportement physique des matériaux mous dans divers contextes, tels que les films d'animation. Ce type de déformation est toutefois connu pour être difficile et fastidieux à reproduire manuellement par les artistes, et les outils existants restent limités pour une utilisation artistique.

En pratique, ces déformations sont donc générées par des méthodes de simulation physique. Cependant, en raison de leur dépendance temporelle, elles doivent être exécutées après les étapes de rigging et d'animation, ce qui empêche une édition non linéaire de la scène 3D. De plus, les artistes ont souvent recours à des effets de déformation caricaturaux pour mieux transmettre les émotions et les idées qui sont difficiles à obtenir par simulation physique.

La principale contribution de cette thèse est un nouvel outil de déformation purement géométrique et indépendant du temps qui assiste l'artiste en résolvant les contacts locaux entre les objets élastiques, ainsi qu'en produisant des effets de gonflement qui peuvent être contrôlés par l'artiste. Pour parvenir à une intégration transparente dans le processus de création d'animation, nous avons conçu notre outil de déformation de manière à fournir un retour instantané à l'artiste tout en permettant une édition non linéaire grâce à une stratégie entièrement indépendante du temps. Pour produire des effets de gonflement plausibles, notre méthode peut aussi préserver intégralement le volume, bien que des contrôles artistiques soient également possibles pour explorer des comportements plus exagérés. Plus précisément, à partir de plusieurs maillages en intersection, notre déformeur calcule d'abord les parties des surfaces restant en contact, puis applique un déplacement procédural contrôlé par une courbe de profil. Même si notre outil traite chaque image indépendamment, il réalise des déformations temporellement continues avec un contrôle artistique du gonflement grâce à un petit nombre de paramètres de pseudo-rigidité. La plausibilité de la déformation est encore renforcée par la répartition anisotrope du gonflement préservant le volume. Une extension est également proposée pour gérer les auto-collisions entre des parties adjacentes d'un même objet, qui se produisent fréquemment dans le contexte d'animation de personnages.

Le résultat de ce travail est un déformeur temps réel robuste qui permet de gérer des configurations géométriques complexes telles qu'une balle écrasée par une main, des lèvres qui se touchent, des doigts qui se plient, etc.

Mots-clés : Animation 3D, deformation de maillages, resolution de collision, informatique graphique, traitement de la géométrie.

Remerciements

Comme dans un célèbre long-métrage d'animation de Disney j'ai eu trois bonnes fées pour m'accompagner dans cette aventure qu'a été ma thèse : Pierre, Gaël et Pascal. Merci d'avoir cru en moi quand je n'y croyais plus. Merci de m'avoir supportée dans tous les sens du terme tout au long de ces quatres années, de m'avoir rattrapée plus d'une fois au bord de l'abandon et d'avoir été là, chacun avec vos spécificités, vos sensibilités, à chaque instant de mon doctorat.

Je voulais également remercier mon jury : Damien Rohmer et Maud Marchal pour avoir accepté d'évaluer mon manuscrit, Loïc Barthe pour avoir présidé mon jury et enfin Stefanie Hahmann pour avoir accepté de faire partie de ce jury et pour m'avoir donné goût à la géométrie et à la modélisation depuis mes études à l'ENSIMAG. Je retiendrai votre bienveillance et vos remarques dans vos rapports et durant la soutenance, ainsi que les discussions intéressantes qui m'ont amenée à réfléchir sur d'éventuelles perspectives pour mes travaux.

Merci à mon guide de SIGGRAPH et de Los Angeles en général, Valentin, pour ta gentillesse, ta bienveillance et tes conseils pour arriver au bout de cette thèse. J'espère pouvoir venir de te revoir une fois que cette pandémie sera derrière nous.

Ces quatre années n'auraient pas été les mêmes sans l'équipe MANAO. Merci à Anne-Laure de m'avoir rendu la vie plus facile, de m'avoir aidé pour toutes les tâches administratives ou pour les missions. On n'oubliera jamais le dossier MITACS ! Merci à Romain et Patrick d'avoir veillé sur moi pendant le premier confinement. Vos messages du vendredi soir me faisaient chaud au cœur et me faisaient me sentir moins seule pendant cette période. Romain, je n'oublierai pas les tablettes de chocolat noir à la fleur de sel et les Dinosaurus non plus.

Cette thèse a également été marquée par de belles rencontres avec les non permanents de l'équipe avec qui j'ai partagé cette aventure et quelques bières au *Stag and Loar* : David, Charlie, Charlotte et Corentin. Je me rappellerai toujours de ces soirées de confinement à jouer à AmongUs. Ne croyez jamais Corentin s'il vous dit qu'il n'était pas dans la trappe ! Même si nous ne nous sommes pas vu souvent, merci Alban de m'avoir soutenue tout au long de cette dernière ligne droite, ces derniers mois qui m'ont paru interminables. Ces longues discussions du soir en jouant à Code Names (ou pas) m'ont aidé à garder la tête hors de l'eau pour pouvoir atteindre la ligne d'arrivée. Mégane, comment ne pas te remercier tout particulièrement. Nous avons passé ces quatre années ensemble, du début à la fin, et tu as toujours été là pour me remonter le moral quand ça n'allait pas. Je me souviendrai de ces après-midi passées à discuter alors qu'on prévoyait à chaque fois de courtes pauses. Nous en avions toutes les deux besoin pour évacuer, reprendre notre souffle et repartir au travail. La seule chose que je regretterai c'est que

nous n'ayions pas pu aller ensemble à SIGGRAPH cette année présenter chacune notre papier. Si l'on nous avait dit au départ que nous publierions toutes les deux en même temps à SIGGRAPH nous ne l'aurions jamais cru. Je crois que nous pouvons en être fières !

Et enfin, *last but not least*, j'aimerais remercier ma famille et mes proches de m'avoir soutenue tout au long de cette aventure. Je sais que cela n'a pas été facile pour vous, mais vous avez toujours répondu présent pour moi. A n'importe quelle heure je savais que d'un coup de fil je trouverai le soutien dont j'avais besoin pour continuer d'avancer.

Cette victoire et ce titre de docteur c'est à vous tous que je le dois, merci.

Publications

The work presented in this manuscript appeared previously in the following publications:

[BBG21] Camille Brunel, Pierre Bénard, and Gaël Guennebaud. A time-independent deformer for elastic contacts. *ACM Trans. Graph.*, 40(4), 2021.

[BBGB20] Camille Brunel, Pierre Bénard, Gaël Guennebaud, and Pascal Barla. A time-independent deformer for elastic-rigid contacts. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(1), 2020.

Contents

1

Introduction

Virtual worlds are parallel universes that allow us to escape from our everyday life. Although imaginary, to be convincing they must refer to the real world that we know through visual cues. In the context of traditional, hand-drawn animation, the animators Frank Thomas and Ollie Johnston theorized in their 1981 book "**The Illusion of Life: Disney Animation**" [TJ81], twelve basic principles of animation in order to produce an illusion that cartoon characters adhered to the basic laws of physics. These principles are still relevant for virtual characters of today's computer animation [Las87].

Many objects in our everyday surroundings exhibit elastic deformations when put into contact with another object: a cat walking on a pillow, a hand pressing on a window or even our cheeks being crushed by our grandmother every time we see her. They most notably squash inside the contact region and bulge as their volume gets redistributed outside of it. The representation of this type of material in virtual world must preserve those desirable properties of the object's appearance and motion to be convincing. Such squashing and bulging effects are essential to communicate plausible deformations. They are particularly important in character animation, for instance when artists set out to convey actions of a character on the environment or on another character (e.g., grabbing, pushing, pressing, etc). As an illustration, Figure 1.1 presents various examples of elastic contacts of character skin in industrial productions.

Over the past thirty five years, digital models and virtual worlds have become an integral part of everyday entertainment. Driven by the development of animated films and video games, the need for virtual object modelling, deformation and animation tools is increasing. More recently, streaming platforms have made animated series and films even more accessible by bringing them directly into our homes. This encourages the development and research of new methods and tools to facilitate the creation of more and more complex virtual characters while stimulating the creativity of artists.

The creation of 3D animations is a complex process requiring different steps and artistic skills. A solid organization and structure are therefore necessary to achieve such a project as the production of an animated film. This creation pipeline is summarized in Figure 1.2. In particular, after all characters and environment assets have been designed by modelers, it is time to bring them to life following the storyboard given by the director. The characters must be equipped with different connected system to be able to come to

Figure 1.1.: Contacts in animated films. Contacts are everywhere in animated films. Different types of contacts can be observed. On the top left and bottom right collision between a rigid and an elastic objects are illustrated with the hands of Coco on the window and the belly of BayMax pressed by the armor. Top Right: Self collisions are also present especially on the skin of articulated characters, here on the arm articulation. Bottom Left: Collision between elastic objects in general, for example between the cheeks of Bao and the woman.

life: the motion system, the control system and the deformation system. All three are usually gathered under one term: the rigging, which often reduces to skeleton animation. Using these tools, the animator can direct these digital models like a puppeteer animates his/her puppets.

This research focuses on the deformation system and in particular the deformation of elastic objects during a collision in computer animation. The particularities of this field of application impose several requirements on the design of the used tools.

Deformations: artists' needs

As previously stated, in animated films, the resulting animation should be believable. However, as a reflection of the director's and artists' imagination, it is highly art-directed, both in terms of poses as well as deformations. It not only communicates events, but also emotions and thoughts. Unfortunately for the artist, the expected deformations often deviate from what is physically accurate, and they usually depend on the animation

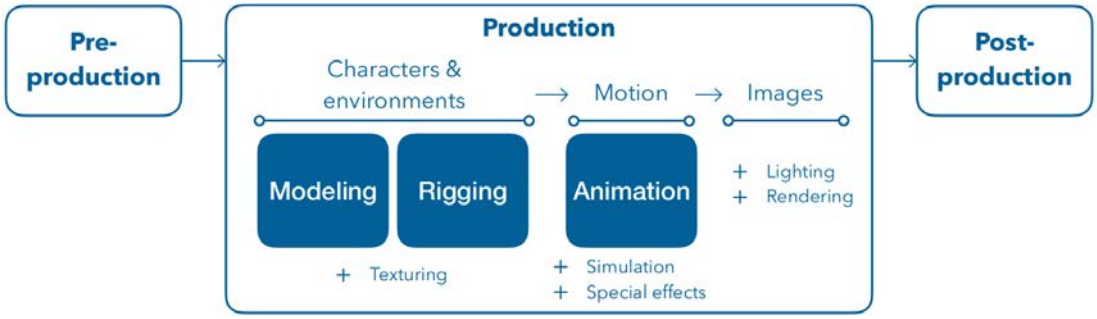


Figure 1.2.: The classical animated film production pipeline is composed by three major steps. The *pre-production* is the preparation stage. It is here that the story, the storyboard and the animatics are designed, but also the aesthetic of the complete universe that the director wants to create. Then comes the *production* stage where all the visual elements needed for the final project are created: 3D models, sets, etc. First the geometry of characters, sets and props imagined and designed in the pre-production stage are recreated in the modeling environment along with the definition of their material visual properties and textures. The outer shell of the virtual models are created but are still lifeless. The rigging stage prepares the characters for animation, it defines the shape deformations and the controls to be used by the animators, like the strings attached to puppets to make them come alive. Then the animators bring them to life like puppeteers following the storyboard conceived in pre-production as reference. Simulation and special effects are used to animate unscripted animations such as fluids, cloth or smoke. These elements just react to their environment with a behavior mainly dictated by physics laws. As in live action films, once all the actors and sets are in place, we can turn on and set up the lights and the cameras and start filming, in this case through image synthesis renderers. The final touches are finally added in the *post-production* stage to make the final production look polished and professional (compositing, color correction, sound effects, etc.). The final product is eventually rendered at this stage.

style and the sought-after aesthetic of the production. Furthermore, the animators lead the viewer to the essence of a situation, for example using exaggeration. For the collision between elastic objects, the squash conveys information about the physical properties of objects (mass, stiffness, etc.) but an exaggerated squash can also be used to magnify the effect of the collision. For all those reasons, we can assume that physical accuracy is not a requirement in computer animation. Physical simulations can create realistic motion but they are poor at stylizing movements as they are constrained by the laws of physics. In this work, we will thus rather aim for *plausibility* to preserve this artistic freedom characteristic of cartoon animation. Nevertheless, as previously mentioned, although not restrained by physics law, to be plausible and believable some visual cues need to be maintained during the deformations. For example, when a ball hits a hard floor, unless it has a hole, it should roughly keep its volume and it is the same for the character skin. Another requirement concerning collision is the production of clean contact. When two objects collide, they should be clearly in contact at some point. The animators even prefer a slight intersection than a remaining gap between the two colliding surfaces.

Finally, to be able to reproduce the desired animation dictated by the directors' will, character rigging and rigging in general attempt to provide animators with as much flexibility and control as possible. Producing an entire animation film requires a large amount of work, as they do not lie on live-action plates as visual effects feature film. On average, an animator can produce from 3 to 8 seconds of animation per day for a film or serie depending on the expected quality. *Artistic control* combined with *interactivity* to not slow down the artists' work are therefore essential features to take into account in the development of a deformation tool in this field.

In the very interesting course proposed by McLaughlin et al. [MCC11], Larry Cutler expose the deformation requirement at DreamWorks Animation that summarize all our goals in this research: high degree of cartoon squash and stretch, stylized character designs, exaggerated and non-realistic range of motion, art-directed results, and immediate feedback and control in the animation environment. We refer the interested reader to this course for an enriching overview of the many challenges related to the task of designing and animating digital characters in film and game production.

Deformations: technical solutions

Elastic objects are notably hard to animate by a computer graphics (CG) artist, especially when they collide with each others, because reproducing their squashing and stretching behavior implies to manually craft plausible deformations in both space and time (e.g., using lattice deformers [NS13]). The classical approach to address this problem relies on the physics of elastic objects, either by explicitly simulating their deformations (e.g., [NMK+06]), or by relying on plausible but faster approximations (e.g., [MCKM15]). The obvious advantage of these approaches is their physical accuracy, provided artists manage to find the physical parameters that yield the sought-for behavior. In practice, such an approach only provides to the artist an indirect control of the elastic behavior and requires extensive training and skills, and frequent and time-consuming trials and errors even for a simulation expert. Moreover, it does not easily allow the exaggeration of the deformation, which is commonplace in cartoon animation. Besides, the major limitation of simulations in an animation context is their dependence on time: the simulation needs to be run from the first frame to the current frame of the animation every time the artist modifies a parameter or the input animation. Therefore, due to their time-dependency, physical simulations must be run after the rigging and animation steps, preventing non-linear editing of the 3D scene.

The alternative solutions, at the opposite end of the methodological spectrum, are manual, fully artist-controlled approaches such as those based on blend shapes, free-form [SP86] or pose-space deformers [LCF00]. Their main advantage is their simplicity: they provide instant feedback to artists, who are then in charge of producing compelling deformations. In practice, bulging effects remain scarce in production because the task of sculpting deformations and animating them by hand requires a significant amount of time, even for accomplished artists. Even worse, each deformation is specific to the shape of objects and their actual contacts, hence it cannot be reused in different situations (e.g., from shot to shot).

For character animation, an in-between class of solutions [MZS+11, GMS14] produces time-independent deformations using quasi-static simulations — even though distant contact deformations still depend on the path taken to the colliding state. These approaches can handle more general deformations than purely geometrical deformer, but they are much more computationally demanding due to their iterative nature. Moreover, these methods offer few control to the artist and, since they are based on physical laws, they cannot produce exaggerated deformations typical of cartoon animation. Geometric modelling approaches handling collisions (e.g., [LB19]) can be applied in this context, but they require expensive optimization and lack artistic controls of the deformation.

The main objective of this thesis is to develop a deformation tool that assists the artist by managing contacts and bulging effects in an art-directable way. As previously mentioned, a seamless integration in animation workflows requires: (1) that the tool provides instant feedback to the artist; and (2) that deformations are time-independent to allow non-linear editing. For plausible bulging effects, it is also desirable that the method preserve volume to some extent; even though artistic controls should also be possible to explore exaggerated responses.

Only a few deformation techniques provide both interactive and time-independent solutions. However, they also exhibit important practical limitations. Procedural deformers, such as the one proposed by [Wan15] or those integrated in industrial software like Autodesk Maya© plugin *iCollide* or the Cinema4D© *collision deformer* offer instant deformations with control of the bulge profile but are limited to the asymmetric case of a collision between a rigid and an elastic object. They focus on interactivity and artistic control and ignore volume conservation, which often yields to odd deformations. Finally other methods provide (rather time-consuming) approximations of volume conservation and are restricted to specific configurations, such as the neighbor joints of a rigged character [VBG+13], or the deformation of a cage that approximates the elastic surface [APHS11].

Our approach

In this thesis we propose a geometric, surface-based and time-independent approach to resolve local contacts and self contacts between elastic objects, producing plausible and art-directable bulge deformations. We focus on local deformations as opposed to global ones that are expected to be managed by the animators through control structures like the skeleton in the case of articulated characters. For example, in the case of a character who slaps the face of another one, we will aim at handling the deformation of the cheek resulting from the collision with the hand but not the displacement of the head if the slap is too strong.

Our idea is to rely on the computation of two regions on each object involved in a collision: the *contact zone* which corresponds to the part where the two objects in collision will stay in contact, and the *deformable region* that will be smoothly deformed to counterbalance the volume initially enclosed by the surface in intersection. The extent and shape of the regions are controlled by few, simple parameters, and the resulting deformation can be computed instantaneously at any frame of an animation. Yet, the produced

shape is stable during the animation without introducing any temporal dependencies. The deformation is continuous in space, time and even when the relative stiffness between the colliding object changes. Even though plausible deformation are obtained using the "basic" approach with few parameters, we also explore artistic control to design the shape of the bulge such as its amplitude, distribution, or the addition of higher frequency details. This way, the user is free to produce either plausible deformations or stylized ones to convey exaggeration or other cartoon effects.

Manuscript outline

The structure of this manuscript mainly follows the different steps of our deformation pipeline.

In a first part, after reviewing the existing work on collision detection, which is the first essential step in the definition of contact, Chapter 2 will give an overview of related methods in the domain of elastic deformations, from physical simulations to manual approaches. An emphasis will be put on the volume preserving and collision-free techniques. Approaches specifically designed for articulated characters will be finally reviewed.

An overview of our approach in Chapter 3 will then introduce our contributions through the presentation of the different steps of the general pipeline developed in the context of this thesis and some of its alternatives. All these steps will be more thoroughly described in the following chapters.

Our method, working on each frame independently to fulfil the time-independence requirement, is based on two main steps described in Chapter 4 and Chapter 5. In a first stage detailed in Chapter 4, we resolve the collision and determine the *contact zone*, that is the region of each involved mesh that will remain in contact with the opposite one at the end of the deformation. The second stage, presented in Chapter 5, corresponds to the computation of the deformation resulting from the collision around the previously defined contact zone. The entire deformation, either in the contact zone or in the surrounding region, should comply with our objectives of plausibility and controllability. To this end, multiple artistic controls will be introduced throughout these two chapters and an interactive painting tool offering the possibility to the user to spatially controls the volume and distribution of the bulge will be detailed at the end of Chapter 5.

An extension of our pipeline for the application to articulated skinned characters will be presented in Chapter 6. This type of application requires special treatments, especially around joints when the two considered colliding parts correspond to adjacent regions on the same initial surface.

Chapter 7 will be dedicated to the presentation of numerous results generated with our method. Comparisons to existing methods will be proposed to evaluate the plausibility of the produced deformations, and the extent of the artistic control possibilities offered by our method will be illustrated. Of course, our method comes with some limitations presented through corner cases configurations. A performance study and potential optimizations will also be discussed at the end of this chapter.

The work presented in this thesis deals with only one of the many physical effects found in computer animation and resolved today using time-dependent simulation techniques that lacks artistic intuitive control. In Chapter 8, we will conclude this document and discuss the opening of new perspectives for future research with the same philosophy of giving to the artist more control to design plausible animations and even exploring more cartoon effects without being limited by the laws of physics.

2

Related work

We will start this chapter by a quick overview of the state-of-the-art collision detection techniques. In a second section, we will present the full spectrum of methods used to handle contacts of deformable objects.

1. Collision detection

Collision detection is used in many domains of computer science: robotics (e.g., for path planning), physical simulations (e.g., surgical simulation) or even character animation. The goal of this section is not to exhaustively review this field of research which is beyond the scope of this thesis, but we will still strive to give an overview of the main families of methods. Moreover, we will focus on the class of methods that are the most related to our application: i.e., exact collision detection between polygonal meshes. Note that we are not interested in collision response neither. For more details on this broad topic, we refer the reader to the following surveys: [TKH+05, YKH+10, Eri04].

Broad vs. Narrow phase. In a generic 3D scene, since any object can potentially collides with any other, performing an exhaustive pairwise collision test would be a nonstarter because its quadratic cost would make it too computationally expensive given that it must be performed at every frame. This observation is even worse when considering object primitives (e.g., triangles). Therefore, to reduce the computational load, collision detection methods can be classified in two phases [Mir97]: the *broad phase* and the *narrow phase*.

On the one hand, the broad phase, also called n-body processing, aims at reducing the number of pair tested by identifying smaller groups of objects that may be colliding and by quickly excluding those that are definitely not in collision due to their distance.

On the other hand, the narrow phase, also called pair processing, is in charge of determining the exact collisions between pairs of objects or primitives. Sometimes the detection of collision between primitives is called the *exact phase*, but it is usually included in the narrow phase and will not be discussed here. Collision detection can thus

be seen as a pipeline of successive filters. These two phases can be performed using different approaches to optimize the detection process.

Discrete vs. Continuous collision detection. In many applications we only need to know if a collision occurs between two objects at a given point in time, a frame of an animation for example. In such a case, discrete collision detection techniques are used. These techniques have the advantage to be "fast" but they can miss collisions if an object jumps past another one between two time steps: this is called the *tunnelling effect*. It can cause problems for path planning for example if we want a robot to avoid any obstacle, or in cloth simulation untangled state.

To alleviate this issue, continuous collision detection (CCD) methods have been developed. For example Bridson et al. [BFA02] proposed a CCD technique for cloth simulation using linearly interpolated trajectories. These methods identify the first time of contact and are thus more accurate. However, they require a long computation time and are not widely used in interactive applications. Many approaches have been proposed to reduce the cost of CCD methods, for example with feature-based bounding volume [HF07] or normal cone tests [TCYM09, WLT+17]. In our case, as we want a time-independent tool, we will focus on discrete collision detection.

Without aiming for comprehensiveness, we will cover in the next three sections some of the classical optimization techniques proposed in the literature to speed up discrete collision detection. First, we will describe different spatial data structures and how they can be used to narrow down the collision tests efficiently. We will then present another family of methods that uses discretized representations of the input data to detect collisions. The last section is dedicated to additional optimization techniques, some of them can be combined with the two previous categories.

1.1. Spatial data structure

Instead of performing expensive primitives intersection tests directly, the idea here is to group these primitives (or objects) inside simpler geometric shapes to perform early, cheaper overlap rejection tests. If two bounding shapes do not overlap, their respective embedded group of primitives do not intersect. Moreover, to avoid the quadratic worst case scenario mentioned earlier, when a large number of objects are considered for collision, they should be divided into small disjointed subsets to speed up the tests. We will present two ways of representing this partition.

Bounding Volume (BV). A bounding volume corresponds to a simple geometry that encapsulates a more complex object or a set of primitives. They are used to improve the efficiency of overlap rejection test. Different types of bounding volumes have been proposed, as illustrated in Figure 2.1. Spheres [Hub95] have been widely used as they are easy to implement, have simple and relatively fast intersection tests, and are invariant to rotation which is important in animated environments. However, they are not a good fit for all shapes like, for example, long and flat objects. Axis Aligned Bounding

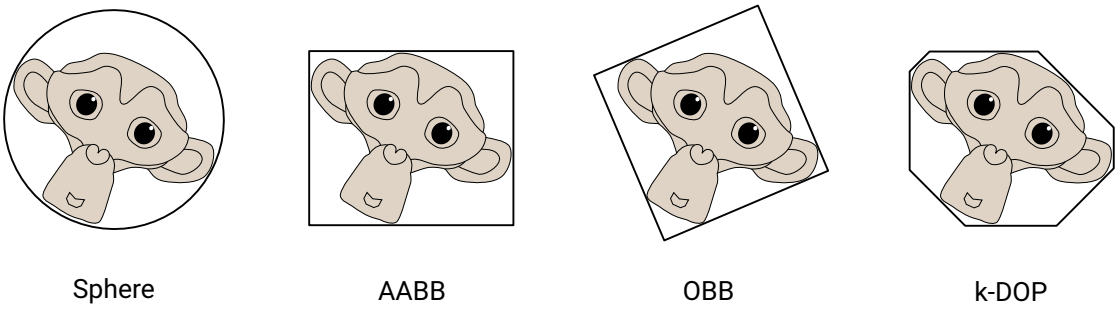


Figure 2.1.: Bounding volumes. Examples of the different types of the most used bounding volumes on a 2D object. They are ordered from left to right, from the coarser but most computationally efficient to the tightest but most expensive in terms of overlap rejection test.

Boxes (AABB) [Ber97] are even more computationally efficient in terms of overlap test, and they are a better fit in general than bounding spheres. However they require dynamic updates during an animation since they are not rotationally invariant. Oriented Bounding Boxes (OBB) [GLM96] have a better fit than the ABBB and Spheres, and they are also invariant to rotations. The counterpart of the quality of the fitting is their higher complexity making both their intersection tests and construction significantly more expensive. Last but not least, let us also mention k-sided discrete orientation polytopes [Zac98, KHM+98], or k-DOP, which are a generalization of AABB in 3D. Indeed an AABB is a 6-DOP. They require the same update when the included objects or primitives are rotated. It is worth noting that larger k values improves the fitting approximation quality but at the cost of more expensive intersection tests. The main challenge here is to find the best trade-off between the number of tests (one for each side) and the speed of each test. More complex shapes fit more tightly the underlying geometry and thus fewer overlap tests are needed but each test is more expensive.

Bounding volume hierarchies (BVH). Bounding volume hierarchies arrange any of the previous bounding volumes into a tree structure to reduce the number of tests that need to be performed by efficiently culling blocks of irrelevant object parts. The internal nodes of the tree contain BV information and its leaves contain object primitives. BVH can be used to accelerate both the broad and narrow phases processing.

The construction algorithms of these trees can be classified into three primary categories:

- *top-down*: This approach starts with the root node containing all the scene primitives. At each step, the primitives are split into two or more disjoint subsets that correspond to the node's children, and they are bound in the chosen bounding volume. Those are further processed recursively. The recursion continues until a termination criterion (e.g., maximum tree depth, minimum primitive count per leaf) is met.

- *bottom-up*: This method starts at the primitives level, with the input set as the

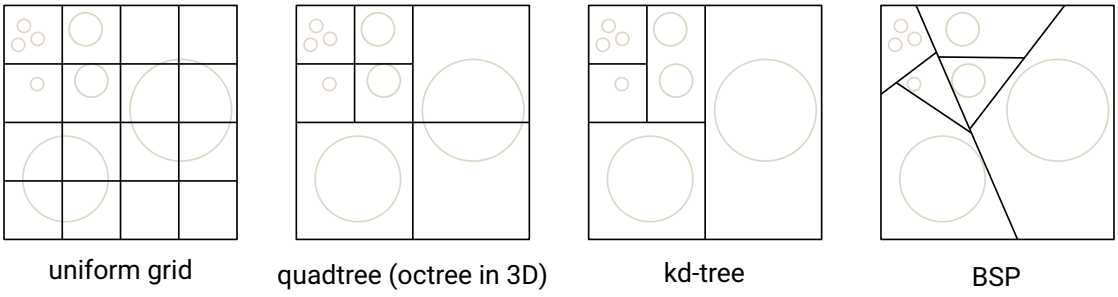


Figure 2.2.: Spatial partitioning. Examples of different spatial partitioning structures in 2D for the same input scene.

leaves of the tree and then group two or more of them to form a new internal node, bounding them in the chosen bounding volume. It then proceeds in the same manner until everything has been grouped under a unique rote node.

- *insertion*: This approach starts from an empty tree and then all other primitives and their BVs are added one at a time to this tree minimizing the increase of volume.

Top-down methods are the most commonly used because they are the easiest to implement and the fastest [LGS+09].

For animated objects, the BVH needs to be either rebuilt from scratch, updated, or simply refitted while keeping the tree structure. Deformable objects are very challenging and the update cost must be considered for choosing a given BVH type. In general, BVH based on spheres, AABB and k-DOPs are the most well suited for arbitrary deformations, such as in cloth animation, thanks to the simple constructions of both the BV themselves and the BVH. Nonetheless, some methods have been proposed to reduce the update cost of a BVH. For instance, the Bounded Deformation Tree [JP04] is a bounding sphere hierarchy which can perform collision detection of reduced deformable objects (i.e., represented as a linear combination of displacement fields) at costs comparable to collision detection with rigid objects. Larsson et al. [LAM03] also proposed a solution in the special case of objects deformed by morphing or blending, building a BVH over one target pose and refitting it to the others.

Space partitioning. Spatial partitioning is an approach to accelerate broad-phase processing. It consists in recursively partitioning the embedding space into disjoint regions and associating the objects (or their bounding volume) with all the regions that they overlap. These methods allow to restrict pairwise object tests to objects that are located in the same region of space. Several spatial subdivisions can be used: uniform grid [Tur90], octree [BT95], k-dimensional tree [Ben75] and binary space-partitioning (BSP) tree [FKN80]. An illustration of these different structures is given in Figure 2.2.

Sibling regions of spatial partitioning structures never overlap, never extend beyond the space of their parent, and their union always covers their parent space. These properties distinguish them from the bounding volume hierarchies. The absence of overlap

can potentially cause memory issues due to the redundancy: if a primitive belongs to two adjacent cells, it is usually duplicated in the data structure. We can also notice, when comparing to BVH, that each node of a spatial partitioning structure can be implicitly seen as a bounding volume. For grids and $k-d$ trees, it corresponds to AABB, for BSP tree to a k-DOP.

In addition to the fact that these techniques assumes finitely size scenes, they have to be rebuilt or updated every time the objects configuration changes, whereas a BVH can be simply refitted. Moreover, the efficiency of uniform grids is highly dependent on the chosen cell size, they can thus be computationally expensive or prohibitively memory intensive. However many methods have been developed to alleviate this issue. For example, Teschner et al. [THM+03] use a hash function for compressing a potentially infinite regular spatial grid to reduce computational overhead, and can detect collisions and self-collisions of deformable tetrahedral meshes in real-time. Computation can also be sped up using Graphics Processing Units (GPU) as in the k-Det algorithm of Weller et al. [WDZ17].

With this type of spatial data structure it is important to consider the balance between overlap rejection cost and memory cost. If the bounding volumes or the leaf nodes (or cells) contain few primitives, only a few intersection tests between primitives need to be performed after the narrow and broad phases, but a large amount of memory is required. Moreover, a tighter tree also implies a more expensive traversal cost to reach the leaves due to either a deeper tree or a more expensive rejection test per node. Conversely, if the cells or bounding volumes embed a large number of primitives, the memory and traversal cost of the structure will be lower, but a large number of expensive intersection tests will be required.

1.2. Discretization approaches

Instead of working directly on the objects, in this second category of techniques, collision detection is performed on discretized representations of the objects or of the space surrounding them.

Image-space techniques. Image-space methods process 2D projections of the 3D objects. This class of approaches is thus convenient to implement on graphics hardware (GPU). They have been applied first for convex objects only [SF91] and then extended to concave objects [MOK95]. They implement intersection tests through the rasterization of the object primitives. Therefore, such techniques do not require any pre-processing, and they directly benefit from GPU acceleration, which makes them especially appropriate for dynamic environments.

Image-space algorithms have been improved using Layered Depth Images (LDI) [SGHS98] such as in the work of Heidelberger et al. [HTG03, HTG04]. The approach of Allard et al. [AFC+10] even uses a layered depth image per dimension to obtain a collision volume, often called Layered Depth Cube. Such methods are obviously also well suited for GPU implementations. However, since the test is performed at the resolution

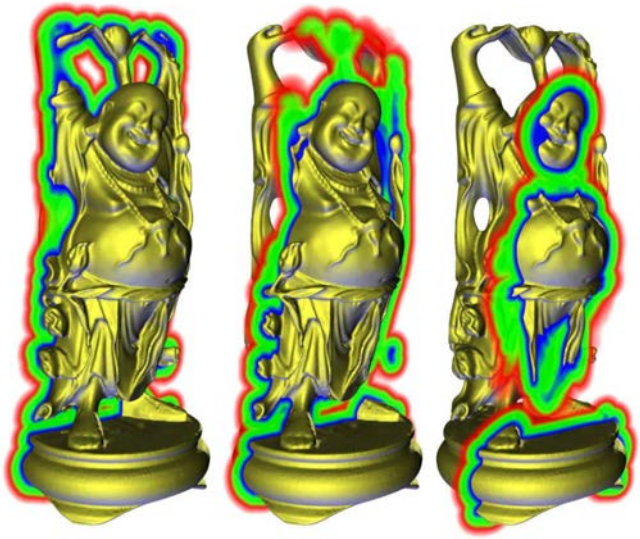


Figure 2.3.: Distance field. Examples of three color-mapped distance field slices from [TKH+05]. The distance is mapped on a color scale from blue for close distances, to green then red when the distance increases.

of the buffers into which the objects are rendered, all image-space collision detection methods are approximate and therefore not applicable in our case.

Distance Fields. Since collisions occur when the minimum distance between two objects is strictly negative, signed distance fields can be used in collision detection (Figure 2.3). They produce robust collision detection, but their bottleneck is the update of the distance fields for deformable object. It can be alleviated for example using image-based approaches [VSC01]. Their memory cost is also an issue since the fields must be stored in a volumetric grid. However this limitation can be improved by only computing fields near the objects and using hierarchical adaptive grids [BMF03].

1.3. Other approaches

Feature- and simplex-based algorithms. Feature-based approaches directly work on the primitives of the considered objects. Their objective is to return the closest features between a pair of objects. This output can be used to determine the closest distance between them, which serves as a basis for collision detection.

The earliest and most well known collision detection algorithm based on the tracking of closest features is the Lin–Canny algorithm [LC91], also known as Voronoï Marching. It consists in partitioning the space around the objects into Voronoï regions that enable the detection of closest feature pairs between polyhedrons. This algorithm exhibits severe flaws that have been addressed by the Voronoï-Clip (or V-Clip) algorithm [Mir98]. However, both methods only work for convex, polygonal, and closed objects. Concave objects must be represented as the union of convex sub-parts.

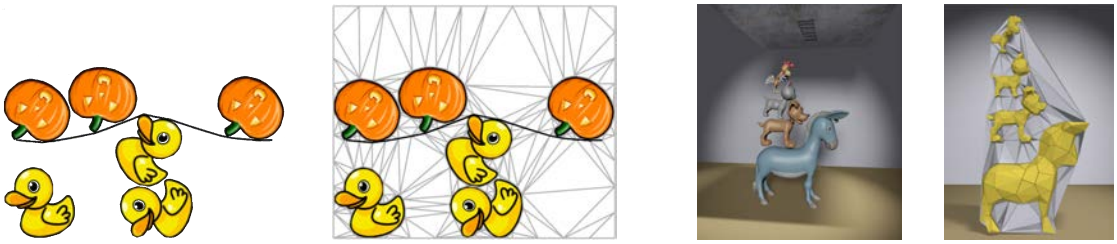


Figure 2.4.: AirMesh. This algorithm considers not the objects but the air between them. Given an input scene with multiple objects, it constructs and maintains a mesh of the empty space between the initial objects. This method applies to both 2D (left) and 3D (right). (Source: [MCKM15])

Rather than focusing on primitives, simplex-based algorithms work on the convex hull of an affinely dependent set of points. The origin of these approaches is the GJK algorithm [GJK88] that uses Minkowski difference on polyhedrons. Two convex objects collide if and only if their Minkowski difference contains the origin, and in that case a measure of their penetration is available. If the origin is not enclosed, the distance between the two input simplex corresponds to the distance to the origin of the Minkowski difference polyhedron. Unlike feature-based methods, since the GJK algorithm does not consider primitives but points sets, it is not limited to polygonal surfaces. For instance it can process implicit surfaces. However, it only works for convex polytopes.

Empty space tracking. Instead of considering the objects themselves, a last family of approaches focuses on the empty space between the objects. A recent example is the *Air-Mesh* method of Müller et al. [MCKM15]. It uses a triangle mesh (resp. tetrahedral mesh), to model the air between deformable 2D objects (resp. 3D objects), as illustrated in Figure 2.4. A collision is detected when the area of a triangle (resp. volume of a tetrahedron) is negative. This method is fast and enables the computation of collision detection and response at once. However, in 3D, the update of the tetrahedral mesh is computationally expensive, thus restricting this approach to configurations for which updates are not required, such as objects moving toward each other like cloth layers.

Culling. Applicable either in the broad or narrow phase, culling approaches eliminate unnecessary tests between objects or object parts based on some high-level properties such as, for example, object velocity [VJ94]. For self-collisions, Barbic et al. [BJ10] use pre-computed certificates which, if satisfied, prove the absence of self-collisions. To be effective the computational cost of a culling approach must be lower than the cost of the eliminated tests themselves.

Self-collisions are a special case for most of the methods presented previously. They are commonly neglected for rigid bodies, but they often need to be considered for deformable objects. Contrary to inter-collisions, which correspond to collisions between two distinct objects, self-collisions refer to collisions between two regions of the same deformable object. Their detection requires a significant computation time as many adjacent or nearby primitives of the deformable surface are very close to each other and can hardly be eliminated by spatial data structures. Bounding volumes can be used to detect self-collisions but different heuristics have been proposed to avoid large overhead due to unnecessary self-collision test of neighboring bounding volumes [VT94]. For a complete review of (self-)collision detection techniques for deformable objects, we refer the interested reader to the recent survey of Wang and Cao [WC21].

2. Deformation of elastic objects

Starting with Lasseter's discussion of squash and stretch in 3D animation [Las87], and the seminal work of Terzopoulos et al. [TPBF87] on elastic models, the quest for physically plausible deformations of soft bodies led to a large body of techniques. Their spectrum ranges from accurate physical simulation (Section 2.1) to geometrical approaches (Section 2.2), some of which are entirely manual. In Section 2.3, we detail how these families of approaches have been applied to the specific case of articulated characters.

2.1. Physical simulation

The classical approach to handle soft bodies deformation in computer graphics is to rely on the physics of elastic objects by explicitly simulating their behavior. Physically-based simulation produces realistic deformations by discretizing Newton's laws of motion and determining the corresponding forces that must be applied to the system. Then the positions of the deformable model are updated through integration schemes. This type of methods can generate complex motions such as, for example, the stretching of an elastic body or the pressure and viscosity of a fluid, but also secondary motions such as the jiggling of a fat belly. They can even handle collision and self-collision adding collision response forces to the system for example.

The obvious advantage of these approaches is their physical accuracy, provided artists manage to find the physical parameters that yield the sought-for behavior. As illustrated in Chapter 7, Section 2.1, many parameters must be provided to setup a simple simulation: the sampling resolution of the input objects, the solver and its time-step, physical quantities such as the stiffness and damping, etc. The combination of all those parameters makes the result of the simulation hard to predict, especially for an artist. Therefore, in practice, setting a physically-based simulation requires extensive training and skills, and frequent trials and errors even for simulation experts. In addition, physical simulations can suffer from instabilities depending on the chosen integration method and time-step. Moreover finite element simulation methods, often used in physical simulation, is computationally expensive, especially when small details must be captured.

For more details about this family of methods, we refer the reader to existing surveys, for instance [NMK+06, MSJT08]. However note that, unlike computational physics, for most applications in computer graphics (special effects, games, interactive systems...) the speed and controllability of the simulation are as important as its accuracy, and visual plausibility is often sufficient. We can distinguish two large classes of methods: the well known ones based on meshes, and meshfree approaches which are usually based on a moving least squares discretization [AOW+08]. For instance, in this later category of approaches in computer graphics, Pauly et al. [PPG04] propose a method handling local deformation of quasi-rigid object resulting from collision. It focuses on local collisions by explicitly computing the contact surface between objects and distributing the traction forces that act on their surfaces to drive a rigid body simulation. In the following paragraphs, we focus on approaches more specifically tailored to real-time applications.

To allow interactive animation of deformable models, *Position-Based Dynamics* (PBD) [BMM15] approximate physical simulation by modeling dynamic particles with masses whose motion is governed by a set of generally non-linear constraints. The system of constraints is composed of equality (bilateral constraint) or inequality (unilateral constraint) equations. For example, collisions can be modeled as unilateral constraints. Solving this system permits to directly update the particle positions, hence avoiding to model external and internal forces. PBD has the advantage to be unconditionally stable and fast solver can be used. However, this method comes with some limitation. First, like all physically-based methods, it is time-dependent. Second, the stiffness of the object is hard to predict as it not only depends on the user parameters but also on the chosen time step which is little intuitive.

Projective Dynamics [BML+14] extends PBD to treat constraints in an implicit manner with a local/global optimization. It avoids requiring constraints to be infinitely stiff. This method results in more accurate deformations than PBD but, despite its performance, it is still time dependent.

In summary, physical simulation encompasses many approaches (mass-spring systems, finite elements methods, position based dynamics, etc.) that make different trade-offs between speed and accuracy. The recent work of Li et al. [LFS+20] even allows the user to specify a target accuracy while ensuring intersection- and inversion-free deformation. Nevertheless, these methods share the two same major limitations: time-dependency and little intuitive artistic controls. These constraints make simulation techniques difficult to use by animators while interacting with 3D objects and prevent their use at rigging and animation stage.

2.2. Geometrical deformation

On the opposite end of the methodological spectrum, the alternative solutions to physically-based methods are the geometrical deformation techniques.

2.2.1. Cage-based space deformation

Cage-based deformation techniques, as detailed in the survey of Nieto and Susín [NS13], fit within the category of purely manual approaches. These space deformation methods consist in the deformation of the 3D space embedded in a cage of control points. Full, direct artistic control is offered through the manipulation of these control points. The space and therefore the enclosed finely detailed mesh located inside the cage are then deformed according to the new positions of the control points and the chosen cage-to-model relationship.

In its simplest form, the cage is a regular control lattice, as illustrated in Figure 2.5. This type of structure is used in the Free-Form Deformation (FFD) technique introduced by Sederberg and Parry [SP86]. To deform the space, thanks to the axis-aligned structure of the lattice, this method defines the deformation of the embedded object using trivariate Berstein polynomials, but other interpolation techniques could be used (e.g., B-Splines). This was the first type of cage that has been used because of their simplicity while producing smooth global deformations regardless of the complexity of the object. Hahmann et al. [HBB+12] use this type of approach to develop a volume preserving FFD algorithm while proposing a GPU implementation. However, these lattice-based approaches suffer from some drawbacks. Indeed, they never perfectly fit the embedded object and the design of a targeted deformation could be complex, for example in the case of articulated character with several limbs.

To better fit the embedded shape, one can define a cage as a low-resolution abstraction of the object, thus enabling to deform it through a simpler mesh, as illustrated in

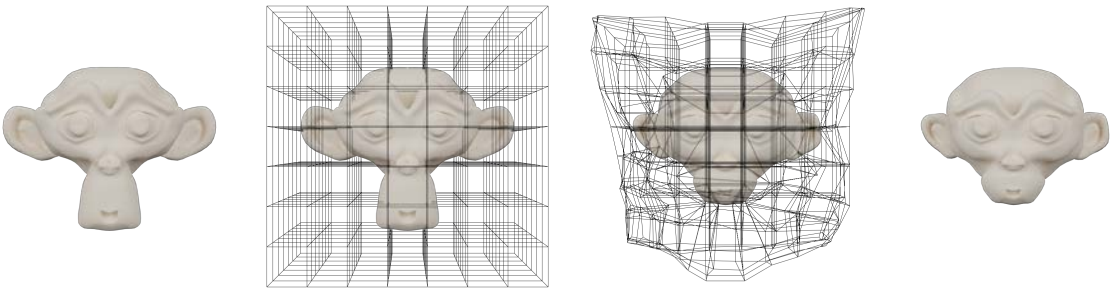


Figure 2.5.: Free form deformation. Right: The input mesh is embedded in a uniform 3D lattice. Left: The manipulation of the vertices of the cage deforms the enclosed space and hence the mesh. (Result obtained using the lattice modifier of Blender 2.91)

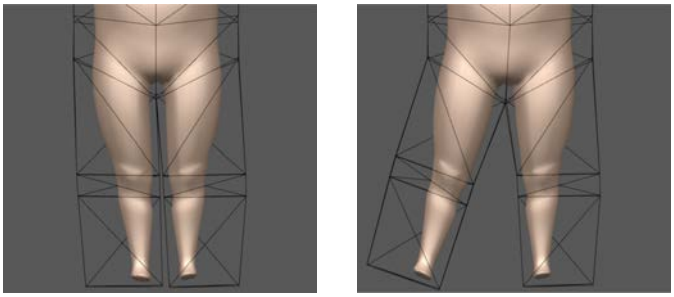
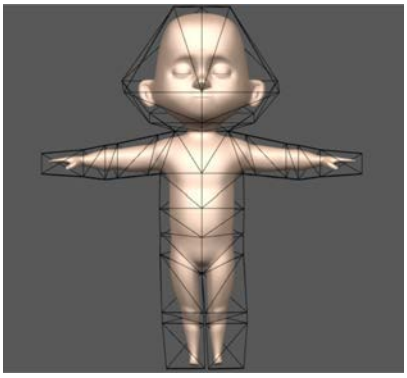


Figure 2.6.: Cage-based deformation. Contrary to FFD, the embedding structure fits each limb of the character separately allowing to deform them individually, here using Harmonic Coordinates to transfer the deformation of the cage to the embedded mesh. (Source: [JMD+07])



the inset figure, from [JMD+07]. Because these spatial structures are potentially irregular and not aligned with the canonical axises, it is not possible to use the simple interpolation scheme of FFD to control the embedded object. As shown in Figure 2.6 several coordinates systems have thus been proposed to satisfy various geometrical constraints during the deformation, e.g., Mean Value Coordinates [Flo03], Harmonics Coordinates [JMD+07], etc. Although automatic methods for generating the embedding cage exist (e.g., [CB17]), they often need to be hand-tuned by the artist to perfectly fit the object and its articulations. Expertise is required for the creation of the embedding control structures but also to manipulate them to achieve the expected deformation. To create an animation, the artist must keyframe the position of the control points in time, indirectly giving life to the embedded object.

2.2.2. Data-driven approaches

A second group of approaches in the field of geometrical surface deformation is the one of data-driven techniques. The concept behind data-driven approaches is to define a set of sculpted example surfaces, i.e., explicit examples of how the input shape should look like under some given poses, and then to interpolate between them smoothly to obtain the desired intermediate deformations.

A well-known method falling in this category is Shape Interpolation, or Blend Shapes. It is a linear model in which the individual basis vectors are not orthogonal but instead represent individual sculpted key shapes. There are as many dimensions as example shapes. Surface vertices, or control vertices, are thus simply a linear combination of the

Figure 2.7.: Face Blend Shape development modeling for an animated feature realized by David Rutherford (https://mung.artstation.com/projects/l6r2Y)

corresponding vertices on these key shapes S_k:

$$S = \sum_{k=0}^{m} \omega_k S_k,$$

where m is the number of example shapes and ω_k is the weight, also called slider, associated to the shape S_k. Blend Shapes is probably the most widely used approach to skin deformation for facial animation. It allows the artist to deform the face of a character according to example poses corresponding to emotion for example. Each extreme expression is sculpted by a modeling artist like the one presented in Figure 2.7, and the animator just need to linearly interpolate between them to create the final expected deformation. This method also offer a transition between the different emotions. The advantage for the animator is that she/he does not need to animate each part of the mesh precisely and manually. Despite its simplicity and effectiveness, this approach exhibits some drawbacks. Because of the linear interpolation, example shapes interfere with each other. Shape Interpolation is more an accumulation than an interpolation. Therefore, when sculpting the example shapes, the artist needs to be careful in designing poses which are as independent as possible in there deformations or effects.Moreover linear interpolation does not always ensure a smooth transition between the poses because a vertex only moves in a piecewise linear way. Finally, one could thought about using this approach to animate the body of articulated character. However, by limiting the control of the shape to the manipulation of weights between the example shapes, the control would be quite complicated and non intuitive for that type of applications.

To reduce the number of dimension and decouple animation control from sculpting, Lewis et al. [LCF00] introduced the Pose Space Deformation (PSD) technique. In this approach, a pose is defined as a sample in a multi-dimensional pose space. These dimensions are for example the degrees of freedom of a character model (e.g., the amount

of bending at finger joints). The artist sculpts the target deformation for each example pose, usually as a correction over geometric skinning. During the animation, the current pose can be encoded as a feature vector defined in the pose space, and the final shape is defined as an interpolation of the sculpted deformation associated with the considered example poses. The interpolation method should define a smooth transition between the different poses.

Despites the appealing artistic control offered by this type of techniques, a tremendous amount of work is required to sculpt all the example poses. Moreover, these approaches are highly dependant of the chosen interpolation method. Indeed, not all the coordinates of the pose space represent a meaningful shape. In addition, the "reachable" configurations are inside the convex hull of the example poses (no extrapolation). If the desired deformation is not within the set of example poses, this implies that the artist must create more example poses to enlarge the space of reachable configurations. Improving the interpolation method is a way to reduce the number of example poses for the same range of deformations, hence limiting the memory cost of the method.

2.2.3. Surface-based approaches

Finally, a third type of surface deformation techniques is the surface-based approaches. Linear variational mesh deformation techniques are detailed and compared in the survey of Sorkine and Botsch [BS08, SB09]. These methods allow the user to deform an initial input surface thanks to intuitive manipulation of few handles defined on the mesh. Different approaches have been proposed, for example the linearization of a physically accurate deformation model coupled with a multi-resolution hierarchy to better preserve surface details, or the representation of the surface using differential coordinates. The methods falling in this last category try to preserve the geometric details of the surface by maintaining as much as possible local differential properties under deformation, for example Laplacian coordinates which describe the mean curvature and normal vector of a vertex. The differential representation is deformed following user constraints, and the final surface is then reconstructed by optimizing a geometric energy. In the case of linear methods, this reconstruction boils down to the resolution of a linear system of equations formulated using a differential operator such as the Laplace-Beltrami operator. These methods are robust, easy to implement and relatively fast, especially for those involving sparse linear systems taking advantage of optimized dedicated solvers. However, to reproduce the physical properties of real-world materials, such methods may require a large number of user constraints.

2.2.4. Deformation tools with contacts

Most of the methods falling into the three previous categories do not ensure collision-free and/or volume-preserving deformations by design, only a few specific methods do.

First, we can mention in the category of space deformation methods, the field-based techniques of Angelidis et al. [ACWK06] and von Funck et al. [vFTS06] that can be seen

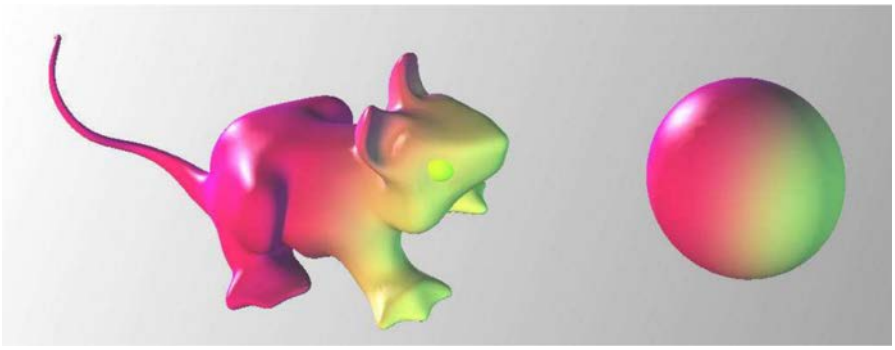
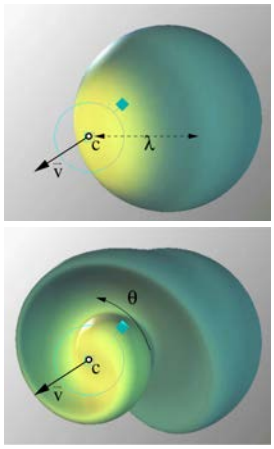


Figure 2.8.: Swirling-sweepers. Example of a mouse model "sculpted" using swirling-sweepers from the input sphere on the right. The two shapes have the same volume. (Source: [ACWK06])

as volume-preserving sculpting tools. For the former technique, the artist draws a deformation path defining the movement of "swirling-sweepers" that causes the deformation of the input shape along this path. As illustrated in the inset, in this approach, a swirl corresponds to a rotational field around an axis $\overrightarrow{v}$ whose angle of rotation θ decreases as a function of the distance from the centre c of the application of this deformation. In particular, the authors show that a swirl deformation is characterized by a deformation whose Jacobian is equal to 1, and thus prove that the method preserves the volume locally without requiring any volume computation. In simpler terms, a swirl twists space locally around a rotation axis without compression or dilation. Moreover, swirls can be combined to create a more complex deformation, as presented in Figure 2.8. However, the user has to choose the appropriate number of swirls to approximate the final deformation and achieving the desired deformation through swirls is rather unintuitive, making it difficult to use in practice. The swirling-sweepers inherit the collision-free deformation property from the "sweepers" introduced in [AWC06]. The motion drags a part of space defined by an influence function, in a manner that prevents the shape from self-intersecting. However this method does not support collisions between arbitrary objects. Moreover, the collision-free and volume-preserving constraints are only ensured for small translation of the tool, otherwise the movement must be decomposed into smaller steps.



The field-based method proposed by von Funck et al. [vFTS06] is based on a C^1 continuous, divergence-free, time-independent 3D vector field. It is used to deform the initial shape by applying a path line integration. The user can manipulate the shape using modeling metaphors like implicit tools, as shown in Figure 2.9. An inner region corresponding to the part that strictly follows the movement of the tool (translation or rotation) and an outer region not affected by the tool must be defined by the user. An intermediate region allows to smoothly deform the shape between the two other constrained regions. The divergence-free vector field ensures volume preservation of

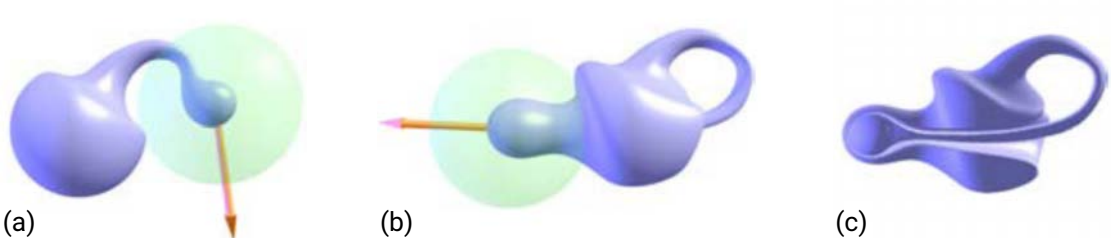


Figure 2.9.: Field-based deformer of [vFTS06]. (a) The implicit tool is dragged by the user to deform an input sphere. The part inside the inner region follows the path of the tool, whereas the part in the intermediate region (in green) partially follows the movement according to the distance to the inner region. (b) Moving the tool through the sphere shape does not produce self-intersection as we can confirm with the cut view in (c). (Source: [vFTS06])

the deformation, and self-collisions are avoided thanks to the path line integration. A specific collision tool is also presented, but this is the only external collision handled by the method. Moreover this method requires expensive numerical path integration. Conceptually this tool is similar to the swirling-sweepers but more flexible since more implicit functions can be used.

In the category of cage-based deformer, Aldrich et al. [APHS11] proposed an iterative method that automatically updates a cage-based deformer to resolve the collision of a rigid object with an elastic one, while approximately preserving the volume enclosed by the embedding cage. Each iteration of the method is decomposed into four steps. First the collisions are detected. Second the cage is updated to locally satisfy the volume constraint. Third, given a target user-defined stiffness, the cage is updated again. Finally the cage deformation is transferred back to the embedded mesh. If collision are still detected, additional iteration are performed. This method provides some control over the stiffness of the elastic object and is free of temporal dependencies, but it only achieves interactive performances with an expensive, iterative GPU algorithm. In addition, since it requires an intermediate volumetric representation, the method leads to global, coarse deformations rather than localized surface bulging, as illustrated in Figure 2.10 where the head is globally deformed by the collision with the rigid plane.

To resolve collisions during shape modeling, Harmon et al. [HPSZ11] proposed a geometric framework based on non-interference constraints on space-time interference volumes. Starting from an initial collision-free configuration, this method determines interference events, corresponding to the exact time at which the collision between two points occurs, and piecewise-linear trajectories of the surface leading to these collisions. From these interference events, a space-time interference volume is computed. The collisions are eventually resolved by deforming the input surface, iteratively minimizing an energy with the constraint that this volume should be zero. This approach is closely related to continuous collision detection methods, and it thus cannot miss any collision, even in the case of thin features. This method also has the advantage of being independent of the deformation and modeling model. However it involves computationally expensive numerical constrained optimization, and does not guarantee volume preservation of the

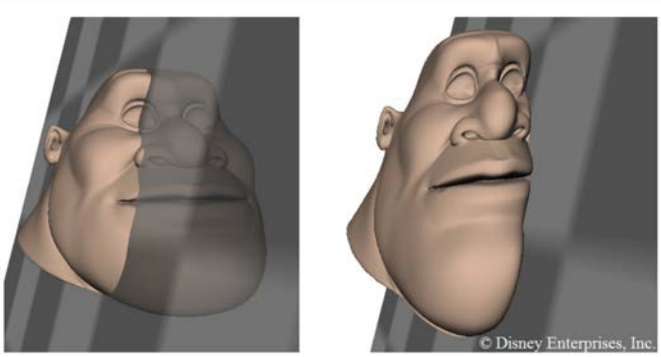


Figure 2.10.: Collision free cage-based deformer of [APHS11]. Deformation of a character elastic head caused by a collision with a rigid plane. Left: original shape. Right: shape after deformation. (Source: [APHS11])

input object. Moreover, artistic control is very limited: only a few editing modes are available, and the artist can only specify whether a surface is allowed to be deformed or not without any stiffness information.

Contrary to the previous method that must be applied as a post-process to a modeling method, Li and Barbič [LB19] proposed a direct collision and self-collision modeling technique. This method is based on the linear subspace deformation technique of Wang et al. [WJBK15] that unifies linear blend skinning and generalized barycentric coordinates to speed up volumetric As-Rigid-As-Possible (ARAP) deformations. This modeling technique produces smooth, geometry-aware shapes from the manipulation of punctual handles and rigid regions. Li and Barbič extend this approach by proposing a multi-resolution and collision free modeling approach where the different levels are activated automatically when collisions or other events require more detailed deformations in a region, as illustrated in Figure 2.11. This method is however restricted to the specific case of handle-based As-Rigid-As-Possible (ARAP) deformation [SA07]. It

Figure 2.11.: Multi-resolution and collision free modeling of [LB19]. Multiple levels, corresponding to the different colored points, are activated around the multiple contact sites (lips, cheek and ear) on the chimpanzee to detect and resolve the collisions. User handles are shown in blue. Right: zoom-in on the resolved collision between the mouth and the hand. (Source: [LB19])

offers a high amount of artistic control for the modeling part, since the user can directly manipulate the surface handles, but a volumetric tetrahedral mesh is needed for the multi-resolution deformation and self-collision detection, and external objects must be provided with signed distance fields for external collisions handling. The collision response is handled by a constraint-based contact model, but no control is offered to the user to control the stiffness of the deformable object or the shape of the response. Finally, as for Harmon et al. [HPSZ11], there is no guarantee of volume preservation.

At last, we explore the work of CG artists who are in the best position to identify their needs and develop the adequate tools. With the assumption that only one of the colliding objects is elastic, procedural deformers, such as the Autodesk Maya© plugin *iCollide* or the Cinema4D© *collision deformer* offer instant deformations with control of the bulge profile. As described by Wang in his Master thesis [Wan15], these deformers proceed in two main steps. First, they detect the points of the elastic object that are inside the rigid object, and project them to their closest position on the rigid surface, hence fully collapsing the intersection region. Second, the elastic surface outside the intersection region is deformed along its normal field proportionally to the interpenetration depth. Since the volume in intersection is not considered, the tool may yield implausible results. Moreover, with this approach the whole pair of surfaces in intersection are kept in contact: as a result, effects as shown in Figure 2.12 cannot be achieved, and instabilities might occur during animations, requiring artists to adjust parameters through time, which is impractical. Nevertheless, such a method meets many of our goals: it is time-independent, art-directable, and it handles contacts while perfectly fitting into the existing rigging and animation pipeline. This is thus an interesting starting point to develop our approach.

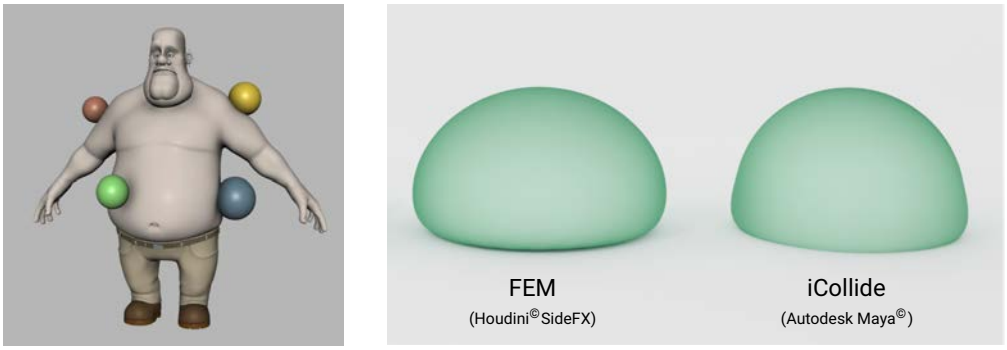


Figure 2.12.: Artists' Procedural deformer. Left: Procedural tools developed by artists themselves, like the one proposed by [Wan15], manage to create plausible deformation, even with multiple contact sites. Right: However, by fixing the extent of the contact surface to the intersection region leads to unrealistic result in some configurations like this elastic sphere colliding with a rigid plane. (Source: [Wan15])

2.3. Articulated characters

In this section we will focus on articulated characters which are essential in computer animation. Their behavior and the resulting skin deformations are critical to give an impression of living beings. The classical approach to animate characters uses a skeleton-based control structure that drives, more or less directly, the deformation of the character skin. A skeleton is basically a hierarchy of constrained frames located at joints, corresponding to articulations in the real world, linked by rigid bones (i.e., line segments). The artist animates a character skeleton by manipulating the degrees of freedom of its joints (usually rotations angles) using either forward or inverse kinematics.

2.3.1. Geometric skinning

To make the skin (i.e., a 3D surface) follow the skeleton motion, geometric skinning techniques are the most widely used approaches as their are fast, simple to implement and artistically controllable. The idea is to define a binding between the skin of the character and the control structure in order to obtain plausible deformations while animating the control structure.

Rigid skinning. The simplest way to define skeleton-skin binding is to rigidly attach each vertex to a single joint of the control structure. This method assumes the following input data: a rest pose shape, usually a polygonal mesh, representing the external envelope of the character and a control structure equipped with transformation matrices $M_j\,,\ j \in [1, m]$. For an animation skeleton, M_j is the transformation matrix aligning the rest pose of bone j with its current animated pose. The final deformed position $\mathbf{p}'_i$ of a vertex mesh i is then defined by:

$$\mathbf{p}'_i = M_j \mathbf{p}_i,$$

where $\mathbf{p}_i$ is the position of the vertex i in the rest pose. This method is called *rigid skinning*. Although simple, this method results in poor quality non-smooth deformations essentially around articulation. Indeed, as we can observe on our own body, when bending an articulation the skin located in the middle of a bone strictly follows its motion, however the deformation is more complex near the joints, leading us to the intuition that each part of our skin is actually influenced by several bones.

Linear blend skinning. This is the idea behind the most well-known skinning technique called *linear blend skinning* (LBS) or skeletal subspace deformation. It was first documented by Magnenat-Thalmann et al. [MTLT88] to compute the deformation of an animated hand, although it was never properly introduced in the literature. As a sign of its popularity, this method is integrated into most computer graphics software with various names: “smooth skinning” in AutoDesk Maya, “bone skinning” in Autodesk 3D Studio Max, “linear blend skinning” in Blender.

This method requires an additional input data: skinning weights $w_{i,j}$ describing the influence of a given bone j on a given vertex i. A common requirement is that the set

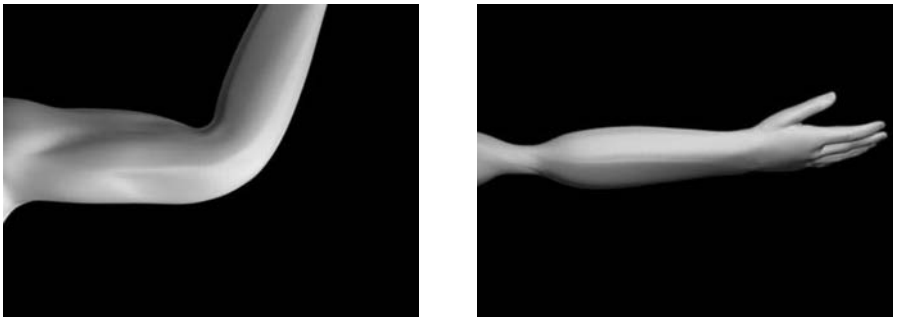


Figure 2.13.: Linear Blend skinning(LBS). This skinning method suffers from two artifacts resulting in a loss of volume: the collapsing elbow resulting in a rubbery appearance of the articulation (left) and the candy wrapper artifact occurring when a joint is twisted (right). (Source: [LCF00])

of weights for each vertex i of the input mesh forms a partition of unity, meaning that $w_{i,j} \geq 0$ and $\sum_j w_{i,j} = 1$. The computation of the deformed position $\mathbf{p}'_i$ of a vertex i is straightforward using the following formulation:

$$\mathbf{p}'_i = \sum_{j=0}^{m-1} w_{i,j} M_j \mathbf{p}_i,$$

with $\mathbf{p}_i$ the position of the vertex i in the rest pose and m the number of bones. The final position of a vertex is thus computed from the rest pose using a linear combination of the transformation matrices of the joints influencing it.

The skinning weight $w_{i,j}$ are most of the time painted directly on the mesh by the user but, due to the partition of unity constraint, the manipulation can become little intuitive for an artist. The manual definition of these weights for all poses of the mesh requires expertise and trial-and-error that can be tedious. Therefore, some methods have been proposed to automatically compute a set of skinning weights such as, for example, the bounded biharmonic weights (BBW) of Jacobson et al. [JBPS11] or the weights defined by Dionne et al. [DdL13] which are determined using geodesic distances from the bones. However, to achieve the expected final deformation, manual fine-tuning is usually necessary.

Despite the production of smooth deformation, this method suffers from two well-known limitations. As illustrated in Figure 2.13, the volume enclosed by the skin can be lost at joints when large rotations occur. This artifact, often called the *collapsing elbow*, can be explained by the fact that a linear combination of rigid transformation matrices does not necessary result in another rigid transformation matrix, for example scaling can be introduced, leading to a volume loss. The second artifact is also related to this linear interpolation, as a linear interpolation of transformation matrices is not equivalent to linear interpolation of their rotation. Therefore the so called *candy wrapper* artifact is especially noticeable when a joint is twisted.

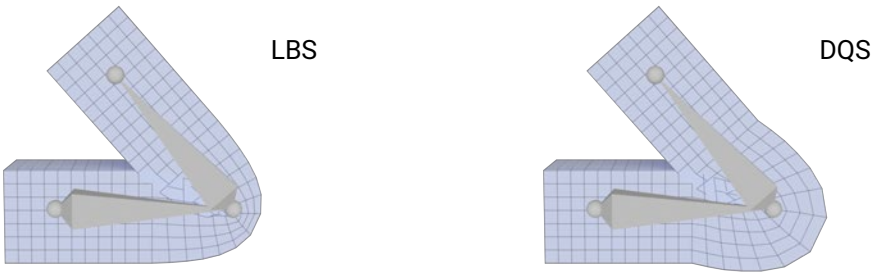


Figure 2.14.: Bulging artifact of dual quaternion skinning (DQS). Contrary to linear blend skinning which exhibits a strong volume loss (Right), dual quaternions skinning produces an unnatural bulge around a bent joint (Left).

Dual quaternion skinning. Many methods have been proposed to alleviate the limitations of LBS and among them non-linear approaches are the most popular. The most well-known method in this category is *dual quaternion skinning* (DQS) introduced by Kavan et al. [KCvO08] with the aim for better interpolation of the the rotation. DQS works similarly to LBS but instead of linearly interpolating the transformation matrices, they are first represented as unit dual quaternions, which are then blended together linearly. Since a linear combination of unit dual quaternions does not, in general, produces a unit dual quaternion, a normalization is performed. The resulting unit dual quaternion can finally be converted to a transformation matrix that can be applied as in the previous approach.

As expected, with proper interpolation of the rotations, DQS gets rid of the candy wrapper and collapsing elbow artifacts of LBS. However it comes with its own limitation: the *joint bulging artifact*, which is reflected as an unnatural bulge around a bent joint as illustrated in Figure 2.14. This artifact is due to the unique constrained center of rotation imposed to vertices moving around a joint. Observing that LBS does not produce bulging while bending, Autodesk Maya allows artists to blend the resulting deformations of linear and dual quaternion skinning, using an additional per-vertex blending weight. The problem with this approach is that even a small amount of LBS may re-introduce the candy-wrapper artifacts.

Elasticity-inspired deformers. Kavan and Sorkine [KS12] avoids this issue by combining LBS and DQS in a non-linear fashion, as shown in Figure 2.15. They define a joint-based swing/twist deformer to better describe the deformation located around the skeleton joints. Specifically, they decompose the rotation of a joint into a swing, i.e., a rotation around an axis in the xy-plane, and a twist component, i.e., a rotation around the z-axis, corresponding to the child bone direction. Then, they apply a 2D spherical interpolation of the twist followed by a linear interpolation of the swing. Since the deformation induced by this joint-based deformer is only accurate in the vicinity of the joint, a linear blending using BBW is then performed between joints to obtain the final result.

This swing/twist deformer requires two sets of skinning weights. The paper describes

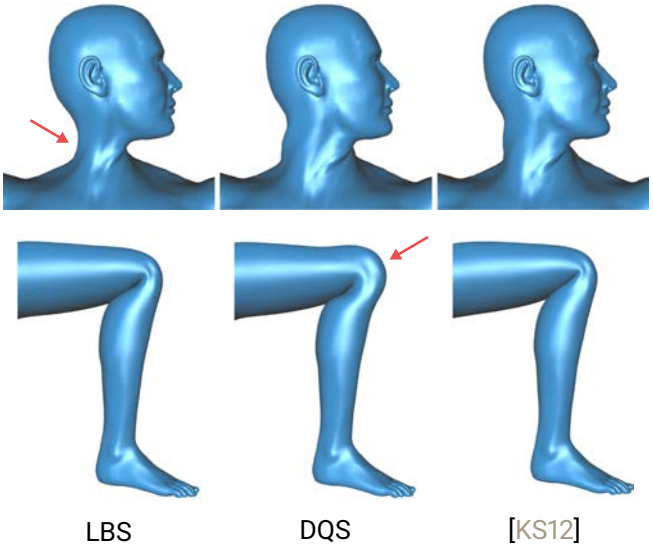


Figure 2.15.: Elasticity-inspired deformers. Thanks to better skinning weights defined using the co-rotated elastic energy and a swing/twist joint based deformer, [KS12] manages to get rid of the candy wrapper artifacts of LBS (top row) and the bulging effect of DQS (bottom row). (Source: [KS12])

an automatic method to compute them so that the resulting deformation minimizes the co-rotated elastic energy over a range of sample poses. It requires a time consuming pre-processing step involving a voxel grid enclosing the rest-pose mesh and a local/global solver to solve this non-linear problem.

Center of rotation skinning. To alleviate the limitations of LBS and DQS, Le and Hodgins [LH16] proposed a direct skinning approach consisting in a correction of the LBS technique through the definition of an individual optimal center of rotation for each vertex of the input mesh, represented with red points in the inset figure. This method is based on the following two assumptions. First, the vertices sharing the same skinning weights should be subject to the same transformation and thus should have similar centers of rotation. Second, local transformations should be rigid.

To meet the first requirement, a similarity function based on the skinning weights is designed. The rigid transformation of the similar vertices is then defined by a rotation matrix which is given here by quaternion linear interpolation (QLERP)[KZ05]. This method tries to find the adequate translation to correct the linear blend skinning error measured using the optimal center of rotation. This correction tends to keep constant the relative distances between similar vertices which limits the collapsing or bulging artifact of LBS, resp. DQS (Figure 2.16).

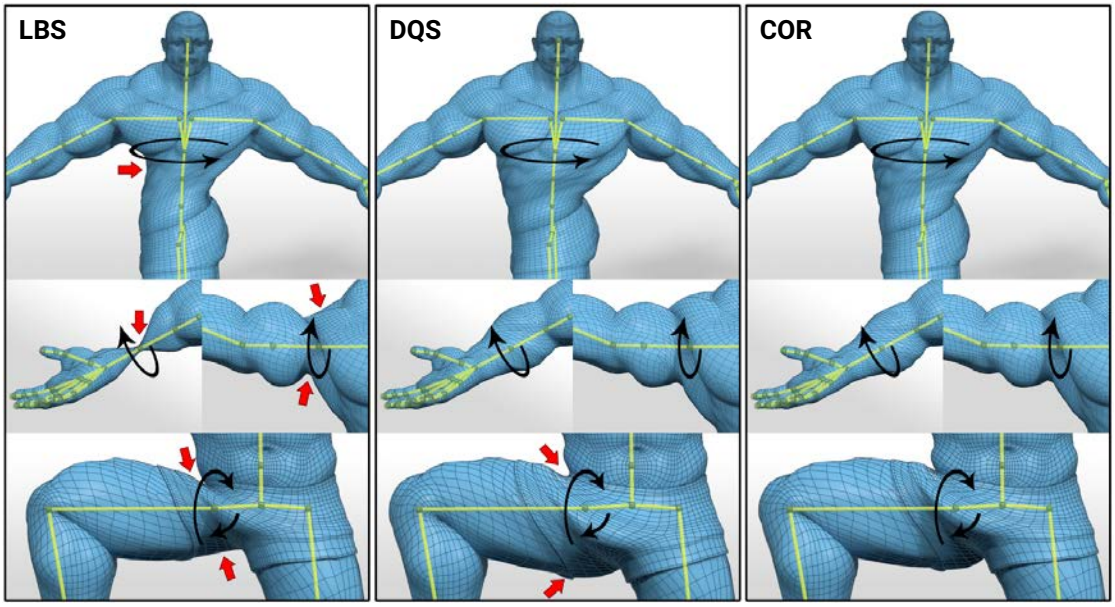


Figure 2.16.: Center of rotation skinning. Comparison on the goliath model of Linear Blend Skinning (LBS), Dual quaternions Skinning (DQS) and Center of Rotation skinning (COR). While LBS suffers from volume loss at twisted and bent joints and DQS exhibits bulging artifact (bottom row), the COR approach manages to alleviate both of these artifacts by defining a center of rotation optimized for each vertex of the input mesh. (Source: [LH16])

The computation of the centers of rotation does not require additional input data compared to the two previous techniques. They are computed as a rather expensive pre-process using the rest pose of the mesh and the skinning weights.

Direct delta mush skinning. The most recent techniques which has been proposed to alleviate the LBS and DQS artifacts while preserving surface details is *direct delta mush* (DDM) skinning by Le and Lewis [LL19]. It extends the previous post-processing algorithm of Mancewicz et al. [MDRW14], providing a direct solution with improved efficiency and control. Conceptually these two approaches apply joint transformations to mesh vertices that have undergone Laplacian smoothing resulting in a "mush". The surface details lost through smoothing are then restored by applying the delta between the smoothed and non-smoothed vertices at rest pose. In addition to limiting the artifacts of the previous methods, this is the first direct method that can produce skin sliding effects. However its storage and run-time computation cost are relatively high: a 4×4 matrix must be stored as skinning weights instead of a simple scalar for other direct skinning methods, and its computation requires one 3×3 singular value decomposition per vertex. Le et al. [LVGO21] recently proposed a compression method to significantly reduce the storage cost and improve run-time performance.

Volume preservation. While trying to produce plausible deformations of the character skin, none of the skinning techniques presented so far guarantees volume preservation after the deformation of the mesh. However, as already mentioned, this is an important visual cue to achieve plausible deformations. Some corrective post-processing methods applied on top of classical geometric skinning techniques have been developed to compensate the loss of volume.

A first method presented by von Funck et al. [vFTS08] consists in the re-injection of the volume lost by the geometric skinning deformation compared to the rest positions. At each frame, the volume correction is applied one joint at a time, starting at the root joint and traversing recursively along the children. For each joint, the geometric skinning deformation is first computed and the volume re-evaluated. The vertices are then moved along an automatically defined vector field in order to exactly preserve the volume enclosed by the character skin.

As noticed by the author, uniform volume preservation is in most cases undesirable. For example, when the arm of a human character is bent, we usually do not want to gain volume on the protruding elbow bone but rather in the bicep region mimicking muscle contraction. A simple heuristic is presented to localize the correction near the joints. If it is not adequate, the user can directly paint positive and negative volume redistribution weights on the surface as illustrated in Figure 2.17.

This approach helps reducing the volume lost by the candy wrapper artifact of LBS, but it does not correct the shape distortions that it produces, and it does not handle collisions or self-collisions. In addition, when applying the volume correction, novel collisions can appear.

Rohmer et al. [RHC09] extend the previous approach by ensuring minimal per-vertex displacement to compensate the volume loss, and by providing more advanced shape controls of the volume correction. These controls take the form of 1D profile curves that describe the geometric distribution of correction to apply along each axis of the local frame of each joint. This approach can thus produce anisotropic deformations, including complex shapes such as folds, as illustrated in Figure 2.18.

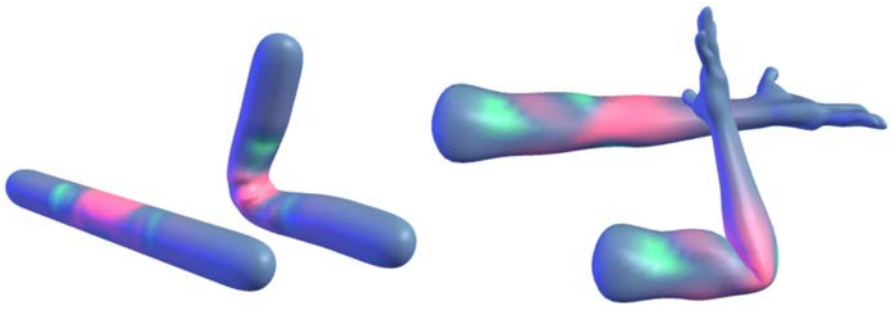


Figure 2.17.: Painted volume distribution of [vFTS08]. Positive (pink) or negative (green) weights can be painted interactively on the surface by the user to design the volume repartition. This can allow to create muscle bulge or folds for example. (Source: [vFTS08])

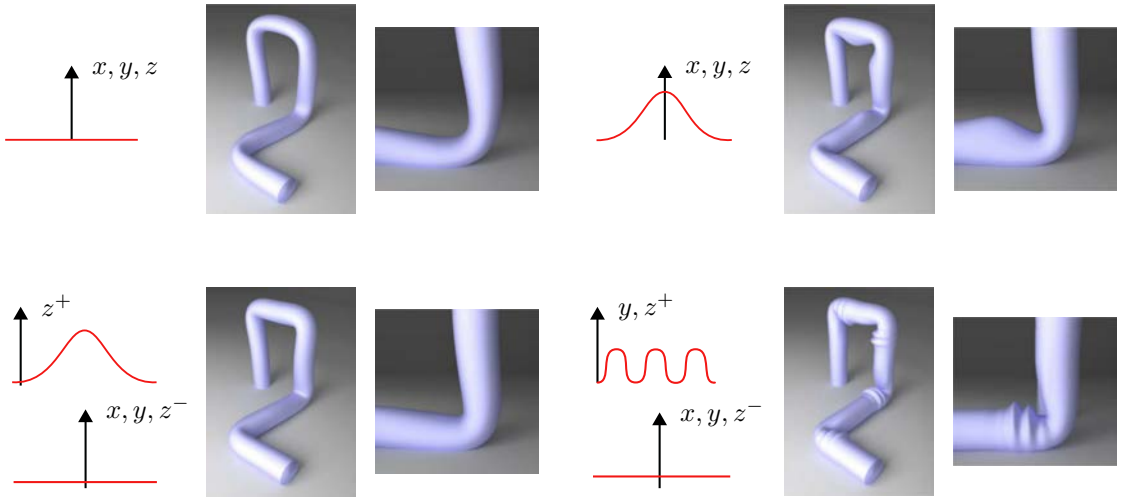


Figure 2.18.: Volume preserving and shape control of [RHC09]. A variety of profiles can be used to preserve volume while offering some local shape control. On bent joints different effects are thus provided according to the volume correction distribution represented by the red profile curves. (a) Reference deformation using Linear Blend Skinning without volume correction; (b) Isotropic volume correction; (c) Biceps-like muscle bulge; (d) Wrinkles effect. (Source: [RHC09])

Although offering more artistic controls, this method comes with some limitations. The final deformation depends on the order of application of the corrections along the local axis as well as the traversal order of the skeleton hierarchy. Moreover, as [vFTS08], this method does not guarantee a collision-free final deformation.

All the approaches presented in this section are purely kinematic and thus lack secondary motion effects such as passive jiggling motion of fat tissues. Moreover none of them produce collision-free deformations by design. And yet both characteristics are known to enhance the believability of an animated living creature. To create more appealing character animations and to resolve collisions, one could consider using pose space deformation techniques where the artists sculpt the contact zone for extreme example poses. This could also offer the possibility to add more dynamic phenomena like bulging muscles. However this process is very tedious.

2.3.2. Physically-based skinning

In order to produce plausible and compelling deformations in a most automatic fashion, many authors have strived to include physically-based models within skinning methods. However, as previously described for general objects, dynamic simulations are intrinsically time-dependent and thus note applicable during the rigging and animation phases.

Quasi-static simulations. By ignoring inertia, quasi-static simulations driven by skeleton motion do not require temporal integration, but are still history-dependent to handle collisions. The work of McAdam et al. [MZS+11] is the first approach that robustly supports large deformations, high-resolution surfaces and accurate collision and self-collision responses. Starting from the animation skeleton and the skin surface, they construct a volumetric lattice embedding the mesh on which the quasi-static elastic simulation is performed. Despite a novel multi-grid solver, it works at best at near-interactive performance, requiring several seconds per animation frame. This method can be extended to support dynamic effects.

Gao et al. [GMS14] introduced an elastic model of the flesh that only includes the boundary vertices (i.e., the mesh vertices) as independent degrees of freedom for the simulation. This surface-centric formulation allows the solver to converge faster than its volumetric counterparts, especially for large meshes, in collision-heavy scenarios. However, it is better suited for linear materials and is restricted to quasi-static simulations. Collisions can be missed for poses that are far away from the training pose space.

Smith et al. [SdGK18] proposed a novel formulation of the Neo-Hookean elasticity, that can be used in the framework of McAdam et al. [MZS+11]. This model maintains the fleshy appearance of the Neo-Hookean model, while exhibiting better volume preservation. It is also more robust to extreme kinematic rotations and inversions. In addition, it is driven by only two physical parameters to directly control the look of the simulation.

To significantly accelerate simulations, subspace methods (also known as model reduction, or reduced-order methods) detect temporal redundancies in the simulation during a precomputation stage. These redundancies, often called training poses, are used to construct efficient, low-dimensional subspaces. Novel simulations are then performed in those subspaces with oder-of-magnitude faster performance. Teng et al. [TOK14] explored such an approach to quickly resolve self-collisions without checking colliding primitives. It can achieve real-time performance, but reduces the computational accuracy and increases the implementation complexity. Collisions can be missed for poses that are far away from the training pose space.

PBD-based skinning. To handle higher resolution meshes at interactive rates while still offering dynamic effects, another solution is to use Position Based Dynamics (PBD) [BMM15]. For example Ruman et al. [ARF15] proposed a method that uses PBD as a correction of the classical linear blend skinning. In a first step, LBS is used to deform the skin mesh and then, using a tetrahedral mesh generated from the skin, they solve a system of geometric constraints using a PBD solver. The constraints are used to model in real-time the jiggling of the skin and to preserve the volume compare to the rest shape. This method manages to overcome the artifacts of LBS but cannot handle collisions or self-collisions. Pan et al. [PCYQ18] extend this approach to handle local self-collisions.

Deul et al. [DB13] combine shape matching of oriented particles and PBD for the simulation of a multiple-layered model, each level respectively representing bone, muscle, fat and skin. To animate the character, the goal position of the vertices of the bone layer

are computed by LBS and distance constraints are applied at each iteration one the vertices of this inner layer. This approach preserves the volume locally by subdividing the initial mesh into sub-volumes and applying the volume constraint on them. Additional position-based constraints are applied to handle collisions detected using discrete collision detection in combination with a BVH. This method offers some artistic control with per layer stiffness parameters. However this method is too slow for real-time skinning and since the definition of the underling tissue depends on skinning weights, so does the result of the method.

Projective skinning. Finally, Komaritzan and Botsch [KB18] propose a two-layered method based on Projective Dynamics, called *projective skinning*. This method takes as input the animation skeleton that will be represented as a volumetric mesh (spherical and cylindrical joints) and the skin surface that will be shrunk onto this volumetric skeleton to define prismatic elements composed by pairs of skin-bone triangles then decomposed into tetrahedron to define a volumetric mesh. The elastic deformation is computed through projective dynamics [BML+14] and anchor constraints forcing the bone vertices to follow the rigid transformation of the animation skeleton. Strain constraints are applied on tetrahedron of the "flesh" to penalize their deformation. This method is thus volume-preserving and produces dynamic effects. However, to achieve real-time performance, they focus on local collisions, global collisions being neglected. Nevertheless, a recent extension of this work by the same authors [KB19] alleviates this limitation thanks to a GPU-implementation of projective skinning that is able to dynamically add and remove collision constraints on demand.

2.3.3. Geometric skinning with contacts

In summary, on the one hand, we have fast and direct geometric skinning techniques, that can be coupled with volume preservation correction methods, but lack secondary motion effects and collision resolution. On the other hand, we have seen physically-based skinning approaches producing more appealing and believable animations but at the cost of time-dependency or computationally-demanding schemes.

Kinodynamics skinning. To try to combine the best of both worlds, Angelidis and Singh [AS07] proposed *kinodynamic skinning* which is based on the observation of von Funck et al. [vFTS06] that a divergence-free velocity field is volume preserving. At each frame, starting from the linear blend skinning formulation, they produce a fold-over free and volume-preserving vector field in pose-space. The positions of the mesh vertices are then integrated over the corresponding path lines from the rest pose to the current pose. Without necessarily being intuitive, manual controls are offered to the artist to control the volume repartition over two given axes, each of them associated with a hand-controllable scalar value, as illustrated in Figure 2.19.

This first part is a history-free skinning technique, but it can also incorporate short-lived dynamic behavior to add secondary effects over a small temporal window. The size

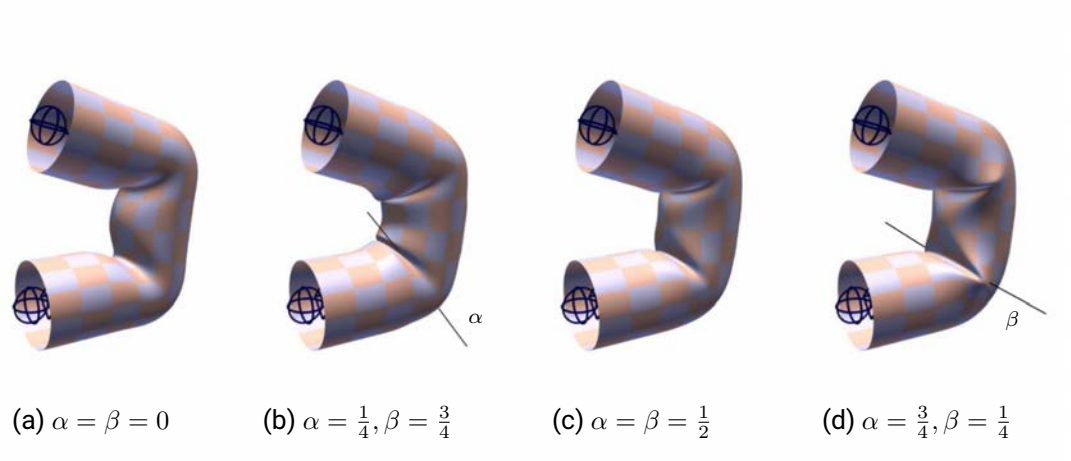


Figure 2.19.: Kinodynamics. While offering volume preservation, various effects can be produced by tweaking two scalar parameters α and β. (a) Volume is preserved when $\alpha+\beta = 1$, while other configurations control the directional incompressibility as show from (b) to (d). (Source: [AS07])

of this window determines how far into the recent past a physical simulation will start (if this size is reduced to zero no secondary effects will be produced). A balance must be found between a large enough window to ensure temporal coherence of the kinodynamic trajectories and a short enough window so that the artist keeps the impression of direct manipulation.

Since solving the whole problem requires some numerical/temporal integrations, it makes the method less efficient on large meshes, and it does not fit very well into the classical animation pipeline. Finally, we can point out that fold-over free deformations do not necessarily mean that a clean skin contact in the folding area of the character limbs will be produced, sometimes gaps may remain, as depicted in the inset figure.

Implicit skinning. Vaillant et al. [VBG+13, VGB+14] address this issue proposing a corrective skinning method. As a pre-process, a volumetric implicit representation of the skin is defined from the character mesh in its rest pose. The mesh is partitioned according to the skeleton bones, and each part is represented by its own scalar field reconstructed using Hermite Radial Basis Functions, as shown in Figure 2.20. To perform the segmentation, each vertex of the mesh is associated to the bone which has the most influence on it, using the skinning weights if available. The parts are then combined together with gradient-based composition operators. The result of this process is an implicit surface which closely matches the initial character mesh, apart from the fine details. To preserve them, each vertex of this mesh is eventually assigned a field value.

During the animation, the field functions are transformed rigidly with the skeleton to

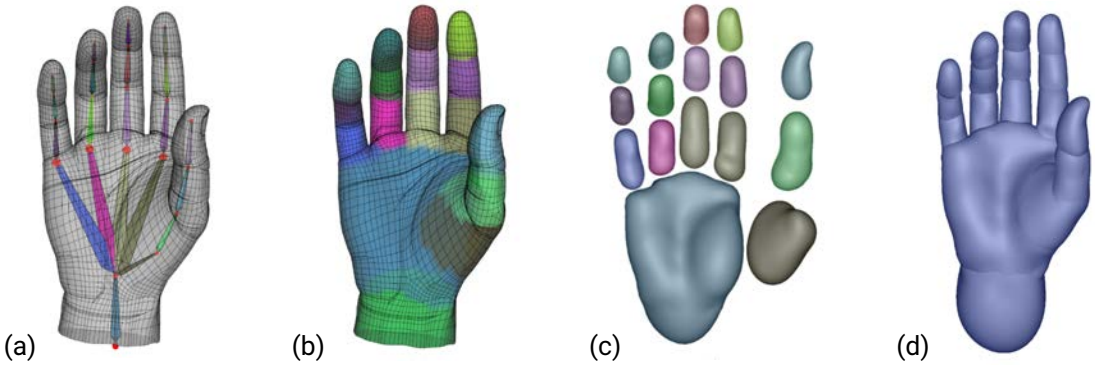
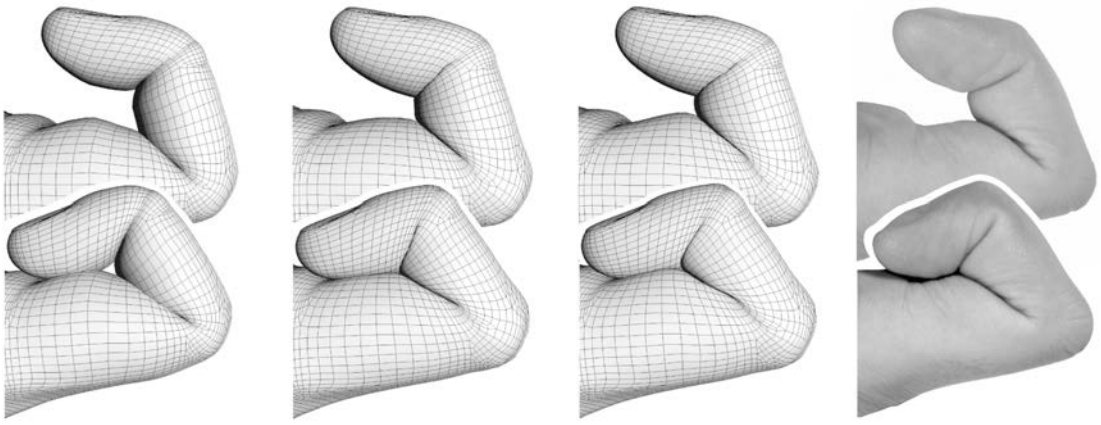


Figure 2.20.: Implicit surface. (a) Initial mesh with its animation skeleton ; (b) This mesh is segmented according to skinning weights. (c) Each partition is aproximated by computing an implicit surface as 0.5-isosurfaces of HRBFs. (d) The resulting shape is determined by composition of these implicit surfaces with an union operator. (Source: [VBG+13])

the current pose, and the implicit surface is obtained by composition of these primitives. The vertices of the skin are independently transformed using DQS. From this initial guess, the vertices are projected on their target iso-surface by marching along the field gradient, and displaced tangentially to minimize a local distortion energy.

Thanks to the implicit volumetric representation of the character, this approach automatically captures contact surfaces between skin parts and handles local collisions of the skin between neighboring articulations without resorting to simulation or requiring any collision detection step as illustrated in Figure 2.21. However distant contacts are too expensive to be handled at interactive rates. Artistic control is offer through the design of the compositing operators, yet gradient-based operators are difficult to art-direct, even though the recent sketch-based modeling tool of Angles et al. [ATW+17] simplifies that process.

This original formulation of implicit skinning is sensitive to the quality of the initial skinning solution, which may lead to artifacts especially when the skeleton shows large rotations with respect to the rest pose. This limitation was addressed in a second version of the algorithm [VGB+14] by transforming the mesh vertices from the previous animation step rather than from the bind pose and by improving the iso-surface tracking with a novel tangential relaxation scheme derived for the as-rigid-as-possible energy [SA07]. These modifications lead to more natural elastic deformations of the skin, but introduce a temporal dependence between frames that makes this method more difficult to integrate into the classical rigging and animation pipeline. Moreover, contrary to Kinodynamics, this method does not guarantee volume preservation. A more recent extension of Rousselet et al. [RARC+18] enriches this model with the addition of muscle dynamics which brings even more realism to the final deformations.



(a) Linear blend skinning (b) Implicit skinning with contact (c) Implicit skinning with bulge (d) Real finger

Figure 2.21.: Implicit skinning. Comparison on a bent finger between (a) Linear Blend Skinning with no contact handling, (b) implicit skinning with the contact operator and (c) the extra volume gained with the bulge operator, and (d) a picture of a real deformation. (Source: [VBG+13])

3

Our approach

As announced in the introduction chapter, our goal is to create a tool to assist artists in the design of deformations resulting from the collision of elastic objects. This tool should be able to run at interactive rates to enable the edition of the deformation at the rigging stage of the animation pipeline. We do not aim for real-time performance, such as in the context of video games with only a few milliseconds available, but rather for a maximum time budget of one second.

Besides interactivity, another desired property is instantaneity. Our method must process each frame of an animation from scratch without any temporal information, but rising to the challenge of preserving the coherence and fluidity of the animation. On the one hand, each frame processed independently should result in a plausible deformation and, on the other hand, the sequence of these frames must remain smooth and jerk-free, without any temporal discontinuities or oscillating motion for example.

In the real physical world, the collision between two elastic objects produces a deformation mostly located around the impact region and whose shape depends on the objects physical properties. As seen in the previous chapter, realism can be achieved using dynamic or quasi-static simulations, but they are not suitable in our context of application as we want instant feedback without any time dependencies. For performance reasons, we restrict our work to local surface deformations: we will not capture global volumetric effects that should be handled by other techniques (e.g., lattice-, cage- or skeleton-based deformations). Moreover, the time-independence objective implies that we must ignore dynamics and secondary motions, such as friction or jiggling for example when the stomach of a character is hit by a ball. Instead, we will focus on the direct, most perceivable effects of the impact between the considered objects: the resolution of their intersection and the redistribution of the volume lost in the collision around the impact zone according to some physical properties of the objects, like their stiffness. In addition, in animation, the quest for physical realism is not mandatory, and sometimes even undesirable, as it may limit the creativity and imagination of the artist.

For all those reasons, we are not aiming for realism but rather for plausibility of the resulting deformation. It allows us to keep the essence of an artistic tool: manual control and a certain amount of artistic freedom. Therefore, our work proposes a automatic solution with volume preservation and a minimal set of control parameters reflecting some

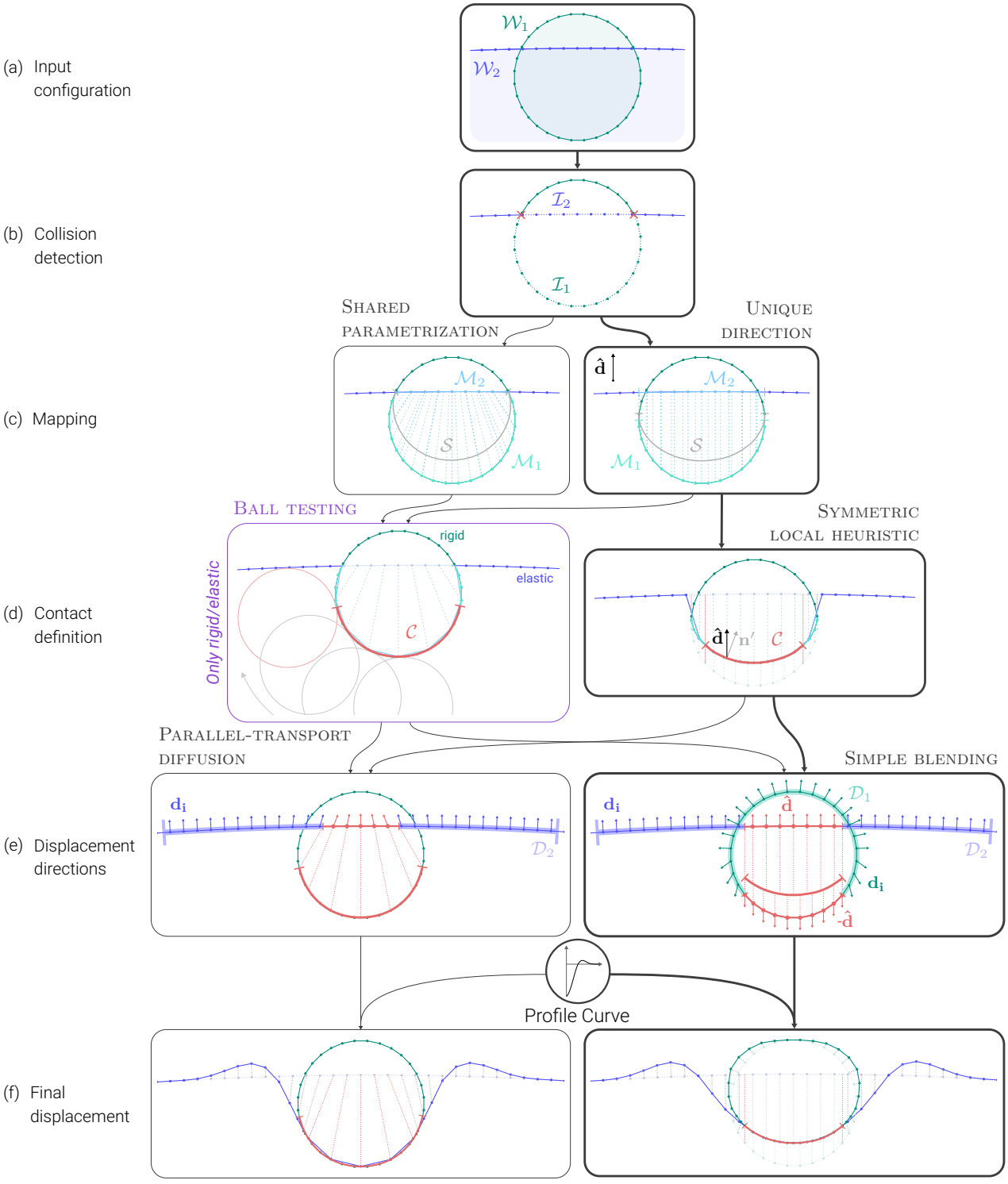


Figure 3.1.: Overview of our approach. The developed pipeline, applied at each frame on each pair of elastic objects, could be divided in six steps described from top (i.e., (a) selection of the working regions) to bottom (i.e., (f) final deformation of the surfaces) here. Our method proposes different alternatives but the preferred pipeline is emphasize in bold.

physical properties of the involved objects, such as their stiffness. To offer even more possibilities to the artists during the creation process, we give them additional controls like volume cancellation or even exaggeration used, for example, to create cartoon effects.

In summary, we want to meet the following four objectives:

- interactivity,
- instantaneity,
- plausibility of the deformation through volume preservation,
- artistic control.

To achieve these goals, we developed a method that processes, at each frame independently, the collision of elastic objects represented as triangular meshes. Our method is based on the observation that a collision between two objects is characterized by an intersection-free deformation that can be decomposed into two parts: a *contact zone*, where the two surfaces will remain in contact, and a *bulge*, in the neighborhood of this contact zone, resulting from the volume displaced when resolving the intersection between the colliding objects. This decomposition allows us to simplify the problem and separate our method into two main stages.

In a first stage (Section 1), we determine the contact surface between the two objects to resolve the collision. We need to determine this region for every mesh that will lie on the final contact surface at the end of the deformation. To be plausible, the shape and position of this intermediate surface must be defined according to the stiffness of the input objects.

In a second stage (Section 2), we compute a bulging deformation around this contact region to give the impression of volume displacement due to the impact. For plausibility, the extent of the deformation around the previously defined contact zone is parametrized again by the physical properties of the objects: the stiffer an object is, the wider this region will be. Contrary to the contact surface which is hidden most of the time (except in the case of transparent surfaces), the final deformation is visible and should thus respect geometrical properties to achieve the goal of plausibility. The deformation should produce a smooth surface with, by default, full compensation of the volume lost by the resolution of the intersection through a bulge designed around the contact region.

For performance reasons, these two stages must use explicit and direct methods, and avoid iterative approaches such as the quasic-static simulation techniques presented in the previous chapter.

Our pipeline is fully illustrated from top to bottom in Figure 3.1. The first main stage dealing with the contact definition is described by the steps (b) through (d) and the last two steps (e) and (f), are part of the final deformation stage. For some steps, we propose different alternatives, but the preferred pipeline is emphasize in bold in the figure. Our method can handle multiple elastic objects, or object parts in intersection, but for the sake of simplicity, we will describe it for a pair of distinct objects, e.g., the blue slightly curved surface and the green sphere in Figure 3.1.

Please finally note that our pipeline can be applied with some additional steps to the case of skinned 3D models. The two considered object parts can then be adjacent on the same surface, for example in the configuration of a bended arm with a collision located in the crook of the elbow. This modified version will be detailed in Chapter 6.

1. Contact definition

As illustrated in Figure 3.1(a), our pipeline takes as input a set of working regions $\mathcal{W}_l$, each $\mathcal{W}_l$ enclosing the expected deformable regions.

Collision detection. At each frame, the first step for handling contact is to robustly detect the intersection between the two considered surfaces. To do so, we perform an edge/face collision detection algorithm over both working regions. This results in a set of regions in intersection $\mathcal{I}_l$ as well as the exact intersection points between the two colliding objects depicted with red crosses in Figure 3.1(b).

Matching. The contact surface corresponds to one or several regions on each colliding object that will remain in contact at the end of the deformation. This observation implies a mapping between the two object surfaces, although only a subset of these mapped surfaces will actually belong to the final contact zone. In theory the mapping and contact definitions are entangled, but to simplify the problem, we first define a conservative mapping between the elastic input surfaces, and then we compute the restricted contact zones. To ensure that the collision is fully resolved, this conservative mapping must include the regions in intersection $\mathcal{I}_l$. The associated projection of those regions onto the opposite mesh is not necessarily inside these regions in intersection and the mapping may thus extend beyond the intersection. The subset of each mesh l finding a match on the opposite surface is part of the so-called mapping regions $\mathcal{M}_l$ depicted in lighter colors in Figure 3.1(c).

We developed two different strategies to compute this conservative mapping. The first one, illustrated on the left in Figure 3.1(c), consists in matching together the intersection regions of the two meshes in parametric space. The resulting mapping directions may consequently potential vary for each vertex of the considered regions. In this version, the mapping regions $\mathcal{M}_l$ are limited by construction to $\mathcal{I}_l$. This method thus suffers from some limitations, such as strong distortions and stretching, discussed in more details in Chapter 4.

To overcome these limitations, we developed a second strategy that matches the two surfaces beyond the intersection regions. We compute such correspondences by finding for each vertex of each mesh within the intersection regions the most distant point on the opposite surface along a unique direction $\hat{\mathbf{d}}$. To avoid considering the entire meshes, we developed an efficient algorithm based on the surface silhouette seen from $\hat{\mathbf{d}}$ to conservatively restrict the computation of the mapping.

Both methods produce a direction field allowing us to project each surface onto the opposite one to resolve the collision.

Potential contact surface. From this mapping, we then determine the potential contact surface $\mathcal{S}$, depicted in grey in Figure 3.1(c). It is located by definition between the two mapped surfaces and its shape and position depend on the ratio of pseudo-stiffness parameters controlled by the user for each object . In the case of the collision between a rigid and an elastic object, $\mathcal{S}$ corresponds to the rigid surface as illustrated on the left of Figure 3.1(c).

As suggested by its name, $\mathcal{S}$ is not the final contact surface but a superset corresponding to the maximal contact surface if both objects were sufficiently elastic to entirely hug the shape of the opposite object. In the next step, we will define the subpart of $\mathcal{S}$, and the corresponding subset of each mapping region, that will effectively remain in contact at the end of the deformation.

Contact zone definition. After projecting the mapped vertices of both meshes onto $\mathcal{S}$, we identify the contact zone $\mathcal{C}$, in red on Figure 3.1(d). Our general approach is motivated by the empirical observation that the stiffer the objects are, the smaller the contact zones should be. Note that, to guarantee the coherency of the animation despite the time-independence requirement, $\mathcal{C}$ must be defined as a region that smoothly evolves on each mesh during the animation. In particular, to avoid jerky movements and oscillations in the animation, we cannot define them on discrete elements such as mesh vertices. Since the contact zone is shared by the two colliding surfaces on $\mathcal{S}$, the projection of the delimited zones on each mesh should result in the same contact zone $\mathcal{C}$ on $\mathcal{S}$. We will define them later as *symmetric*.

Once again, we have developed two different methods for this step. We first propose an algorithm, illustrated in the purple box of Figure 3.1(d), that implicitly takes into account the geometry of $\mathcal{S}$, the depth of penetration and a user-controllable pseudo-stiffness parameter. Inspired by α-shapes [EM94], we translate the stiffness of an elastic object into the radius of a ball rolling on the interior of the surface. The region considered in contact is the one accessible by the ball, in red in Figure 3.1(d). We can observe that the bigger the ball is, the smaller the final region will be, and conversely, the smaller the ball is, the larger the final contact region will be. The radius of the ball thus reflects the stiffness of the objects. The limitation of this first method is its asymmetrical definition, making it only relevant for the case of the collision between a rigid and an elastic object.

Therefore, we developed a second method based on the unique mapping direction $\hat{\mathbf{d}}$. It uses a new symmetric criterion meeting our observations and expectations regarding the contact zone definition. This criterion directly takes into account the geometry of $\mathcal{S}$ through its normal field $\mathbf{n}'$, the depth of penetration and a parameter controlling the extend of the contact zone. This method is illustrated in the right box of Figure 3.1(d).

At the end of this step, we have determined continuous and symmetric regions on each mesh. Once projected on $\mathcal{S}$, they define the interface between the two objects at the end of the deformation.

2. Resulting deformation

After defining the contact surface, in the second stage of our work, we aim at deforming the neighborhood $\mathcal{D}_l$ (in lighter color in Figure 3.1(e)) of the contact zone in a plausible way, independently for each object l. In our work, plausibility will be achieved by satisfying the three following constraints:

- *Smoothness of the deformed surface*: the deformation of the surface should preserve the smoothness property of the initial mesh.
- *Continuity during the animation*: the final animation should be smooth, without jerky or oscillation motions to give the illusion of fluidity and temporal continuity even though each frame is processed independently.
- *Volume preservation*: the deformation should compensate the volume lost by a closed surface in the collision to give the illusion of physical realism.

We consider by default local deformations of closed surfaces requiring volume preservation, but if the object is an open mesh, the user should have the opportunity to partially or entirely overlook this constraint and still obtain plausible deformations. In addition to all these requirements, we recall one of our main objectives, that is, artistic control.

Several approaches could be conceived to compute the final deformation. Inspired by the survey of Botsch and Sorkine [BS08], we could simply define the final surface displacement as a 3D deformation field over the deformable region $\mathcal{D}$ using linear, surface-based algorithms. However, to comply with our geometrical smoothness requirement (tangent constraint at the boundaries) costly biharmonic operators should be used. As-rigid-as-possible (ARAP) surface deformation techniques (e.g., [SA07]) could also be considered, but their non-linear definition is not well-suited in our context, and incorporating volume preservation into their formulation proves difficult. In addition, both approaches lack artistic control over the finale deformation, which is an essential goal in our work.

To simplify the problem and get more controllability over the deformation, our key idea is to separate the definition of the final displacement into two parts: its direction and its magnitude. We define the displacement of the initial surface over the deformable region $\mathcal{D}$ along a smooth unit direction field $\mathbf{d}$. Its amplitude is controlled by a 1D profile curve $\mathcal{H}$ instantiated at every vertex $\mathbf{p_i}$ of $\mathcal{D}$ and evaluated using a one-dimensional radial parametrization u yielding to its final position:

$$\mathbf{p}'_i = \mathbf{p}_i + \mathcal{H}_{a_i,s_i}(u_i)\,\mathbf{d}_i. \tag{3.1}$$

The shape of the profile curve $\mathcal{H}$ is parametrized by the amplitude a_i and slope s_i at $u = 0$. Additional degrees of freedom enable extended artistic control of this curve.

Deformable region. The deformable region $\mathcal{D}$ is defined as the subset of the working region that will be deformed to preserve the volume of the initial object. As previously stated, the extent of this region depends on the stiffness of this object. Moreover, we can observe that it should evolve with the variation of the contact region boundary $\partial\mathcal{C}$.

To both delineate and parameterize this region, we compute a radial parametrization u from $\partial\mathcal{C}$ over the whole working region, and then determine $\mathcal{D}$ following the gradient of this scalar field starting from $\partial\mathcal{C}$. Intuitively, u locates any point of the surface along a smoothed geodesic going from $\partial\mathcal{C}$ to the external boundary of the deformable region $\mathcal{D}$. The extent of $\mathcal{D}$ is determined by a user controlled pseudo-geodesic distance.

It is worth noting that, as $\mathcal{C}$ is changing at each frame, the parametrization u and, by construction, $\mathcal{D}$ must be recomputed dynamically at each frame. Also note that, since their definition is based on $\partial\mathcal{C}$ which is temporally continuous, they also evolve continuously during the animation.

Direction field. The unit direction field $\mathbf{d}$ along which the surface will be displaced is subject to two constraints: (1) it must match the fixed displacements of the contact zone on the potential contact surface $\mathcal{S}$ and, (2) for a smooth shape, it should be mostly aligned with the surface initial normals. We made the observation that, during a collision, an elastic object is deformed outside the impact zone along its normal field, forming a bulge. Therefore the idea is to make the directions converge quickly to the initial normals as we move away from $\partial\mathcal{C}$.

The most natural, but costly, approach to address this problem consists in diffusing with parallel-transport the deviation of the constrained mapping directions from the initial surface normals along $\partial\mathcal{C}$ with the constraint to vanish at the exterior boundary of the deformable region. The resulting unit direction field is illustrated on the left in Figure 3.1(e) by the arrows.

An easier way to resolve the same problem is to diffuse the constrained mapping direction over the whole deformable region $\mathcal{D}$ and to blend them with the normals of the initial surface using a non-linear remapping of the scalar field u. This method is illustrated in the right box of Figure 3.1(e).

Profile curve definition. Concerning the magnitude of the displacement, we define a family of functions, referred to as *profile curves*, instantiated on each vertex of the deformable region. The degrees of freedom corresponding to the amplitude a_i and slope s_i at $\partial\mathcal{C}$ (i.e., $u = 0$) are required to respectively ensure C^0 and C^1 continuity with the contact zone. To be able to evaluate $\mathcal{H}$ everywhere, a and s are diffused over the whole deformable region $\mathcal{D}$. Three other degrees of freedom are needed to guarantee C^2 continuity with the undeformed part of the initial mesh and to ensure that the deformation smoothly vanishes at the exterior boundary of the deformable region. Finally the profile curve should exhibit one last degree of freedom to linearly control the bulge and ensure exact volume preservation.

The main advantage of this definition is that it opens the doors for a wide range of artistic control of the collision response. The bulge can be artistically controlled to exaggerate or cancel the volume compensation. It can even be translated over the parametrization u or spread out to modify the volume repartition. It can also be anisotropically spread over $\mathcal{D}$ to produce more plausible deformation response. We can even imagine producing wrinkles by adding high frequency details to the profile curve.

4

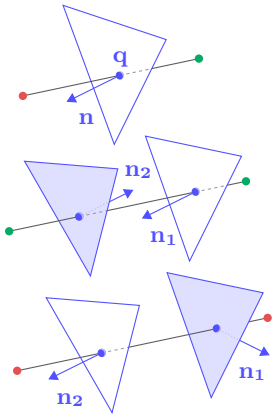
Contact definition

We present in this chapter the first steps of our pipeline that aim to define the regions where two colliding objects will stay in contact after deformation. For the sake of simplicity, as in Chapter 3, we will describe each step for a pair of distinct objects. At each frame, we start from two manifold triangle meshes already in intersection. For performance reasons, we work on a set of open subsets denoted $\mathcal{W}_l$, with $l \in \{1, 2\}$. Those working regions are selected by the user according to the expected spread of the deformation, but it could also be automatically determined as a distance to the intersection region. After detecting the collision between the two involved working regions (Section 1), we compute a mapping between them to define a potential contact surface corresponding to a superset of the final contact surface (Sections 2 and 3). We finally delimit the contat regions on both objects and project them on the shared contact surface (Section 4).

1. Collision detection

The first thing to know when handling contact is whether or not there is a collision between the two considered objects.

At each frame, we test if such a collision exists between the two selected surfaces $\mathcal{W}_1$ and $\mathcal{W}_2$. We also need to robustly detect the regions in intersection, as the definition of the contact surface will rely on them. For each mesh, we perform an edge/face collision detection, illustrated in the inset figure for some typical configurations. We determine the set of edges of the first mesh that are intersected by faces of the other mesh, recording the corresponding intersection points $\mathbf{q}_{k,l}$ in the parametric space of each working region $\mathcal{W}_l$ (i.e., their barycentric coordinates on the intersected edge and face). Note that we neither need to insert these points into the mesh tessellation, nor to chain them together to determine which vertices are inside the collision; we only need to know their positions on both meshes.



Starting from these intersection points, we must now determine which vertices of the initial meshes are inside the region in intersection. For every edge in intersection with one or multiple faces, we consider each of its extremities independently and tag it as either *interior* (green points in the inset figure) or *exterior* (red points) based on the normal $\mathbf{n}$ of its closest intersected face. Considering the closest intersection point is sufficient because it is the only one affecting the tag of the considered edge extremity. Therefore, we search for all the faces intersecting an edge, but we only keep the closest face of each extremity to determine if it is *exterior* or *interior* regarding the opposite surface. These two faces, at most, are sufficient to define the tags of the vertices of the current edge as illustrated in the inset figure. In addition, on the implementation side, it has the advantage that we only need to store at most two faces for each intersected edge.

We eventually propagate the *interior* tags on the vertices of both $\mathcal{W}_1$ and $\mathcal{W}_2$ with a flood fill algorithm to delimit the two regions in intersection $\mathcal{I}_1$ and $\mathcal{I}_2$ (depicted with dotted lines in Figure 3.1 (b)). Since the intersection points are defined by their barycentric coordinates on edges and faces of both surfaces, the intersection regions $\mathcal{I}_l$ evolve continuously over the course of the animation. A discrete definition on mesh vertices would lead to temporal discontinuities as soon as a vertex would enter or leave the regions.

At the end of this step we obtain a continuous definition of the intersection regions $\mathcal{I}_l$ on each object delimited by exact intersection points located on edges and faces of both meshes. These points are stored as pairs of barycentric coordinates corresponding to their location on both surfaces.

Implementation details. We use Embree [WWB+14] in *robust* mode to compute segment-triangle intersections. The computation is accelerated by a 3D Axis-Aligned Bounding Box (AABB) tree built once for each mesh at the selection of the working regions and then updated at each frame if the surface is animated non-rigidly. This update consists in the refitting of the existing boxes without changing the tree structure. If the object is animated rigidly, the corresponding global affine transformation is applied to the structure prior to the search at the new frame.

We did not push our research further in this direction since this step was not the main limiting factor of our approach, but it would be interesting to experiment with more efficient methods, such as those presented in Chapter 2. For instance, instead of computing edge-face intersection from mesh 1 to mesh 2, and then from mesh 2 to mesh 1, one could perform a synchronized traversal of both AABB tree as proposed by the `rtcCollide` routine of Embree.

Discussion. If both working regions are manifolds with boundaries without self-intersections, and if the collision does not overlap these boundaries, then the intersection corresponds to a closed volume and our algorithm produces correct intersection regions. When one of those conditions is violated, the algorithm may fail. For instance, in such configurations, two faces intersecting two different outgoing edges of the same vertex

may give to this vertex an inconsistent tag. In our implementation, the tag given by the last edge considered has the final say, which may lead to a global inconsistent state.

2. Mapping

Once the intersection regions have been determined, we need to define a mapping between the two surfaces. It will then allow us to project each mesh onto an intermediate shared potential contact surface $\mathcal{S}$ and to resolve the collision. We developed two methods to reach this goal. In the first attempt (Section 2.1), we aim at finding a bijective mapping between the intersection regions of the two meshes by matching their intersection points in parametric space. This simplifies the problem, since we can rely on existing correspondances and we can fully work in the 2D texture-space, but it may lead to objectionable artifacts and unnatural mappings. The second method (Section 2.2) addresses these limitations by relaxing the bijectivity constraints between the intersection regions, which allows us to enlarge the mapping regions beyond the intersection regions. Yet, to make the problem more tractable, we must consider that, at a given frame, the relative motion between the two objects is a pure translation along a unique mapping direction.

For both methods, we recall that the mapping region $\mathcal{M}_l$ on each mesh l must be defined as a continuous zone evolving on mesh edges to ensure a smooth definition of the potential contact surface.

2.1. Shared parametrization

Intuitively, to resolve the intersection of the two colliding objects, it seems sufficient to find a mapping restricted to the intersection regions. With such a method, the mapping regions $\mathcal{M}_l$ correspond to the intersection regions $\mathcal{I}_l$ and the boundaries of these regions are by definition the boundaries of $\mathcal{I}_l$, which are evolving continuously during the animation, as described in Section 1.

The key idea behind this method is that we can construct the mapping in parametric space leveraging the constraints provided by the intersection points defined on the edges of the initial surfaces. Here, we assume that both surfaces are equipped with a continuous 2D parametrization. We then proceed in two steps illustrated in Figure 4.1.

First, to minimize the distortion of the final mapping, we compute the global affine transformation, represented by the 3×4 matrix A, that minimizes, in the least-squares sense, the distance between every pair of intersection points $(\mathbf{q}_{k,1}, \mathbf{q}_{k,2})$ expressed in the two parametrizations, i.e., minimizing:

$$\sum_k \|\mathbf{e}_k\|^2 \ , \quad \text{with} \quad \mathbf{e}_k = \mathbf{q}_{k,1} - \mathrm{A}\,\mathbf{q}_{k,2} \ .$$

Through this step, we compute a first coarse approximation of the matching of each pair.

Second, we refine locally this global transformation to enforce the intersection points to match more accurately, i.e., $\|\mathbf{e}_k\| = 0, \ \forall k$. To do so, we compute a smooth 2D

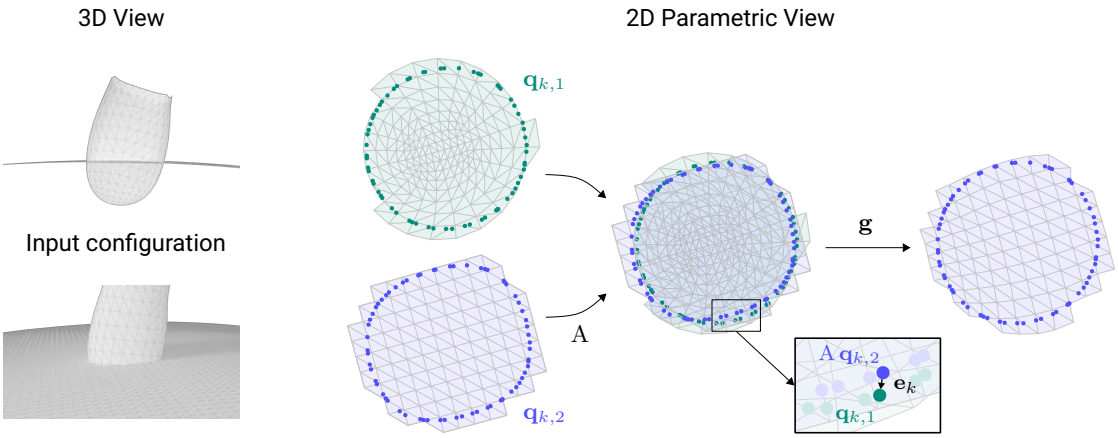


Figure 4.1.: Matching of the shared parametrizations. The mapping between the two surfaces in collision, here a plane and a finger, is performed in 2D parametric space. The regions in intersection $\mathcal{I}_l$ along with the corresponding set of exact intersection points $\mathbf{q}_{k,l}$ are represented in this reduced space. In a first step, we determine the affine transformation A that best aligns the pairs of intersection points. In a second step, we refine the matching by computing a 2D deformation field g through the harmonic diffusion of the error vectors $\mathbf{e}_k$ between two points of the same pair of intersection points.

deformation field $\mathbf{g}$ by harmonic diffusion of the error vectors $\mathbf{e}_k$ over the *interior* vertices [BK04]. As the error vectors $\mathbf{e}_k$ are defined along edges we cannot use standard Dirichlet conditions. Instead, we include the one ring of *exterior* vertices adjacent to the *interior* region and introduce the $\mathbf{e}_k$ targets as linear least-squares constraints as detailed in Appendix B.

At this stage, each *interior* vertex i of one object knows its parametric location in the opposite object parametrization: $\mathbf{q}_{i,1} = \mathrm{A}\,\mathbf{q}_{i,2} + \mathbf{g}_i$. To find its 3D position $\mathbf{p}_i''$ and normal $\mathbf{n}_i''$ on the initial opposite surface, we search the face and barycentric coordinates associated to the final parametric location of this vertex. We perform this association step on both objects. To speedup searches, we use a 2D AABB-tree built in parametric space over the opposite mesh.

Discussion. A first limitation of this method is its asymmetrical definition. Indeed, only one of the two meshes is transformed in parametric space to make the pairs of intersection points match. Swapping one mesh with the other would not impact the affine transformation, but it would change the diffusion which depends on the configuration of the mesh. The resulting mapping will thus not be the same. A solution could be to distribute the error vectors $\mathbf{e}_k$ between the two objects to guarantee a symmetric mapping. It nay also reduce the effect of the distortion on the final mapping described below. The main drawback of this solution is its cost as an additional diffusion would be required.

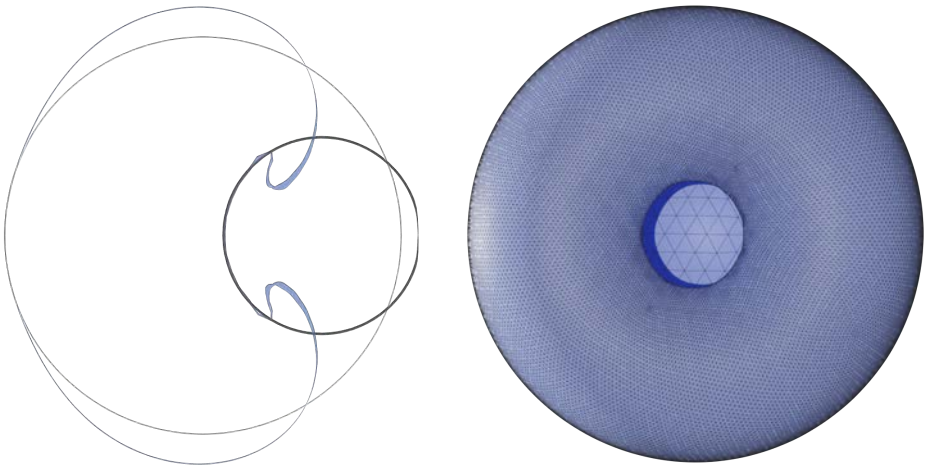


Figure 4.2.: Discretization issue. Left (clipped view): when the rigid sphere in dark grey plunges too deeply into the elastic one in purple, the mapping between the two surfaces inevitably exhibits strong stretching (the undeformed sphere is visualized in light grey). Right (side view): the amount of stretching is better seen after hiding the rigid small sphere.

The second limitation of this approach comes from the bijection between the two regions in intersection. For deep collisions, such a mapping will exhibit strong distortions and stretching, which implies that the potential contact surface becomes under-sampled, which in turn leads to objectionable discretization artifacts, as illustrated in Figure 4.2. This issue is emphasized by the relative stiffness between the two objects. In this figure, the distortion is extreme because the small sphere is rigid and the other elastic. If the relative stiffness between the two meshes were more balanced, the effect would be lessened. A practical solution to this problem would be to refine the meshes in the intersection regions to limit sampling artifacts. Ultimately, it could be interesting to devise an algorithm computing the mapped contact zones without a pre-established parametrization, such that the respective areas of these zones on both involved surfaces are the same. A perhaps less challenging research direction would be to investigate on-demand local parametrizations [HA19] that could be computed in a such way as to minimize stretching around the foreseen contact regions.

Regardless of the distortion, restricting the mapping to the intersection regions is problematic for many geometrical configurations. This issue is directly related to our time-independency goal, as illustrated in Figure 4.3. To reach such a configuration, we know that the foot must have crossed the plane in the previous frames but, since we are processing the animation one frame at a time, we do not have access to this information. Yet, if we think of the potential trajectory of the foot leading to this configuration, we can guess that the desirable mapping region of the plane should extend far beyond the intersection region. Conversely, in Figure 4.2, the mapping region on the small sphere should probably be smaller than the intersection region. We can conclude that the idea of bijection between intersection regions is not suitable for our objective.

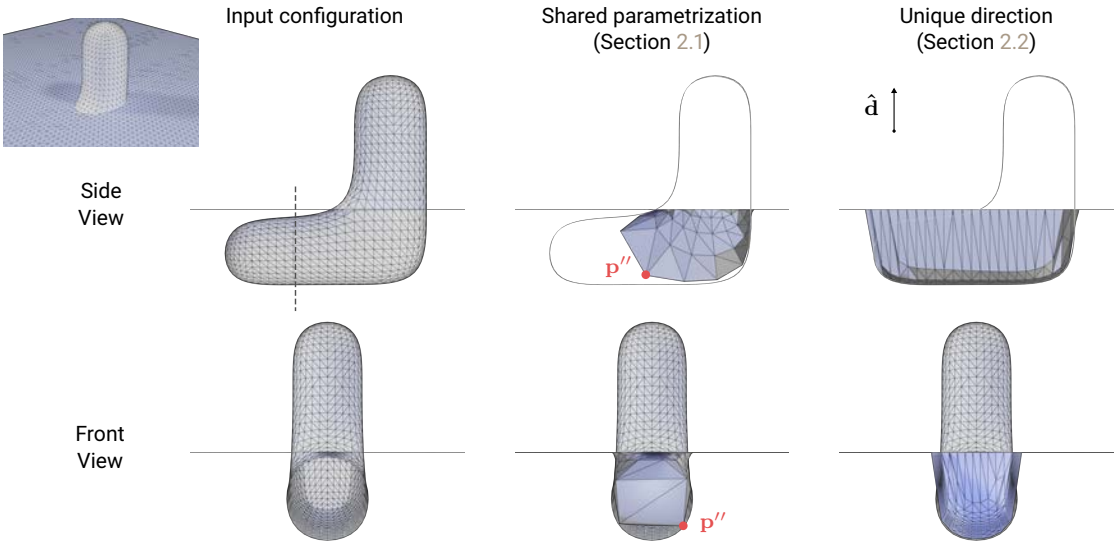


Figure 4.3.: Mapping beyond the intersection regions. Left: A rigid foot going down collides with an elastic plane. The bottom row corresponds to the front view of the configuration cut along the dashed line illustrated on the side view (top row). Middle: The shared parametrization approach produces a poor quality mapping exhibiting large distortions. The side view even gives the impression that the mapping is incomplete, as if $\mathbf{p}''$ was not projected on the foot. Right: Using an unique mapping direction $\hat{\mathbf{d}}$ results in a mapping between the two surfaces that better reflects the apparent motion of the two objects.

2.2. Unique mapping direction

To alleviate the limitations of the previous approach and keeping in mind that the expected contact zones may extend beyond the intersection regions, we developed a second method to compute a partial mapping between the two surfaces. To make this problem tractable in an interactive system, we make the key assumption that, at a given frame, the relative motion between the two objects is a pure translation. It implies that all the points within the contact zones share the same mapping direction $\hat{\mathbf{d}}$, which allows us to efficiently compute mapping regions $\mathcal{M}_l$ tightly enclosing the points that can belong to the contact zones.

However, this assumption has two consequences. First, we ignore sliding effects within the contact region. Second, since the entire working regions have the same direction, we will not be able to handle the case of multiple connected parts that move along different directions of translation. For example, consider a hand grasping a ball: each finger tip touches the ball with a different directions. This limitation can be circumvented by considering each component (e.g., each finger tip) as an independent working region and handling the contact with the ball independently for each of them.Nevertheless, this workaround is only possible if each finger produces a distinct contact, without any interaction with the other ones.

Finally, it is important to note that this mapping direction is defined at each frame and thus can potentially vary over the course of the animation.

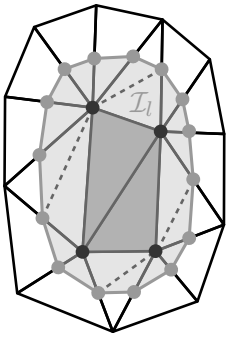
Direction definition. To determine the direction $\hat{\mathbf{d}}$ at each frame of an animation, we propose an automatic method based on the geometry of the two meshes in collision.

For each mesh $l \in \{1, 2\}$, we compute the mean normal $\bar{\mathbf{n}}_l$ over its intersection region $\mathcal{I}_l$. The idea is then to define the direction $\hat{\mathbf{d}}$ as a weighted combination of these two mean normals, giving a higher weight to the object having the smallest variation of normals within the intersection region. Using $1/\mathrm{Var}(\mathbf{n}_l)$, the inverse of the *total variation* (i.e., the trace of the covariance matrix), as weights yields:

$$\hat{\mathbf{d}} = \frac{-\bar{\mathbf{n}}_1 \mathrm{Var}(\mathbf{n}_2) + \bar{\mathbf{n}}_2 \mathrm{Var}(\mathbf{n}_1)}{\| - \bar{\mathbf{n}}_1 \mathrm{Var}(\mathbf{n}_2) + \bar{\mathbf{n}}_2 \mathrm{Var}(\mathbf{n}_1)\|} \tag{4.1}$$

Note that, since the normal fields of the two meshes are in opposite directions, a minus sign is required here. We arbitrarily chose to orient $\hat{\mathbf{d}}$ in the direction of the normal field of the second mesh.

To maintain a flowing animation, this direction $\hat{\mathbf{d}}$ must smoothly vary in time. Let us observe that the intersection region is continuous in time as it is delimited by the exact points of intersection between the two objects (see inset). The temporal continuity of $\hat{\mathbf{d}}$ can thus be obtained by computing the mean and total variation through continuous integrals of a continuous normal field. For the sake of simplicity, we consider the unnormalized vector field obtained through piecewise linear interpolation, allowing for simple Gauss quadrature integration (see Appendix A for details). For instance, the mean $\bar{\mathbf{n}}_l$ is given by:



$$\bar{\mathbf{n}}_l = \frac{\mathbf{m}_l}{\|\mathbf{m}_l\|}, \quad \text{with} \quad \mathbf{m}_l = \sum_{f \in \mathcal{I}} A_f \mathbf{n}_f,$$

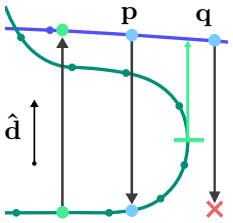
where A_f is the area of the triangle face f, and $\mathbf{n}_f$ denotes its average vertex normals. The considered faces are either faces of the initial mesh (dark grey triangles in the inset), or result from the cut of an initial face crossed by the continuous boundary of the intersection region (grey polyline). In the latter case, the points considered as vertices of the triangle are the vertices of the initial mesh if they are inside the intersection region (black dots) and the exact intersection points otherwise (light grey dots). These cut faces can either be triangles or quads. We split the quads into two triangles (dashed lines) in order to use the same numerical integration technique for the entire region. Using a degree two Gaussian quadrature rule leads to the following approximation of the face normal:

$$\mathbf{n}_f = \int_f \mathbf{n} = \frac{1}{6} \sum_{k=0}^{2} \frac{\mathbf{n}_k + \mathbf{n}_{(k+1) \bmod 2}}{2},$$

with $\mathbf{n}_k$ the normal on the k^{th} vertex of the triangle face f. The normal of the exact intersection points is obtained by linear interpolation of the normals at the extremities of their supporting edge. The same method is applied to compute the total variation.

If the user is not satisfied with this automatically computed direction, she or he can specify it manually and even key-frame it during the animation. For example, this can be used to fake the influence of tangential motion and friction forces. This manual approach is detailed and illustrated in Section 3 of Chapter 7.

Mapping definition. We define the mapping regions $\mathcal{M}_l$ as the set of points (not only vertices) on each mesh that can be mapped to each other. To this end, each object is projected onto the other by matching every vertex with the most distant point on the opposite mesh, when it exists, along the associated mapping direction (inset figure). Note that these points (e.g., $\mathbf{p}$ in the inset) do not necessarily belong to an intersection region $\mathcal{I}_l$, but they are projected onto the intersection region of the opposite mesh. The resulting mapping regions $\mathcal{M}_l$ can thus be larger than $\mathcal{I}_l$. The simplest approach to compute this matching consists in casting a ray along $\hat{\mathbf{d}}$ (resp. $-\hat{\mathbf{d}}$) for each vertex in the working region $\mathcal{W}_l$ of each object and to find its furthest intersection with the opposite mesh. Such an approach would not only be inefficient, since most rays will not find any intersection, but more importantly, it will prevent us to tightly define the mapping regions. If these regions are not conservative, then temporal instabilities will occur when the contact zone reaches the boundary of the mapping region. In contrast, conservatively defining $\mathcal{M}_l$ as the set of faces containing at least one matched point (e.g., the segment $\mathbf{pq}$ in the inset figure) would make further processing (i.e., the construction of the contact surface and contact zones) impossible due to unmatched points (namely, $\mathbf{q}$ which is beyond the projection of the green mesh silhouette onto the blue surface). To solve both problems, we need an efficient algorithm to compute mapping regions which tightly enclose all the points that can be projected onto the opposite surface, and which smoothly evolve over mesh edges to guarantee the consistency of the deformation during the animation. This algorithm proceeds in two steps.



First, we find the smallest region $\mathcal{B}_l$ that needs to be projected onto the opposite mesh to resolve the collision. Based on the observation that the parts of two colliding objects that remain in contact must face each other, $\mathcal{B}_l$ corresponds to the *front-facing* parts of the surfaces in intersection $\mathcal{I}_l$ when seen from their associated mapping direction, as illustrated in yellow and green in Figure 4.4(a). More precisely, defining the orientation function $g = \mathbf{n}^\top \hat{\mathbf{d}}$, a point on the surface is front-facing if $g \leq 0$. These regions are, by definition, delimited by the occluding contours of the surface. To ensure temporal continuity on polygonal meshes, we extract "interpolated" contours for all faces within $\mathcal{I}_l$ using the method of Hertzmann and Zorin [HZ00]. By definition, an edge for which the orientation function has opposite signs at its two vertices i and j is crossed by a contour. The barycentric position t of the contour point along this edge is obtained by linear interpolation of g, i.e., $g(t) = (1-t)g_i + tg_j$; solving for $g(t) = 0$ yields $t = g_i/(g_i - g_j)$.

If the resulting set of contour segments is not empty and forms a closed loop, the region that must be projected on the other surface is simply $\mathcal{B}_l$. If the boundary of $\mathcal{B}_l$ does not form a closed loop (i.e., the *front-facing* region is not entirely contained in $\mathcal{I}_l$),

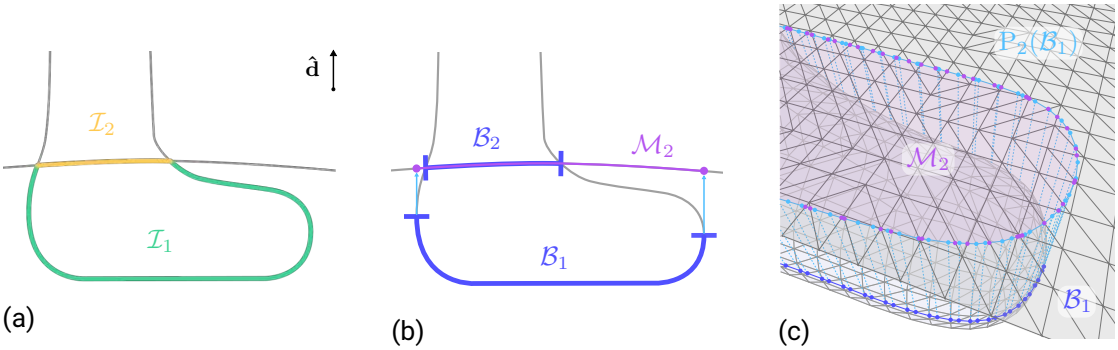


Figure 4.4.: Mapping region definition. (a) The front-facing regions $\mathcal{B}_l$ are computed in the part of the objects in intersection $\mathcal{I}_l$ (in green for Mesh 1, yellow for Mesh 2). (b) The intersection between $\mathcal{B}_1$ and $\mathcal{I}_1$ (blue) is projected on $\mathcal{W}_2$ following $\hat{\mathbf{d}}$. The union of this projected region and $\mathcal{B}_2$ defines $\mathcal{M}_2$ (purple) (c) 3D View of (b). The chained contour of $\mathcal{B}_l$ (blue) is projected on Mesh 2. The intersection between the projected chaining (cyan) and the edges of Mesh 2 defines the exact-boundary point of $\mathcal{M}_2$ (purple points).

we close it by chaining the contour curves with the boundary of $\mathcal{I}_l$ when they intersect. Finally, if the set of contours is empty, it implies that the full intersection region is *front-facing* and $\mathcal{B}_l$ can be reduced to $\mathcal{I}_l$, whose boundary is already continuously defined by the exact intersection points (Section 1).

In a second step, we need to find for each region $\mathcal{B}_l$ the corresponding region on the opposite mesh. In the following we describe our method for the first object; it works similarly for the second one. To obtain continuous regions defined on mesh edges, we extrude the chained boundary segments of $\mathcal{B}_1$ (dark blue ticks and curve in Figure 4.4(b), resp. (c)) along $\hat{\mathbf{d}}$, and consider its intersection with the front-facing part of the other object $\mathcal{W}_2$ (light blue curve). Each intersection between an extruded quad and a front-facing edge of $\mathcal{W}_2$ (purple dots), i.e., such that $g(t) \leq 0$ with t the intersection parameter along the edge, delineates the boundary of the projected region $\mathrm{P}_2(\mathcal{B}_1)$. The set containing all the extruded contour quads is denoted Q. In practice, we work in 2D by projecting all edges and contour segments on a plane orthogonal to $\hat{\mathbf{d}}$. We then perform in this reduced space simple edge-edge intersection tests accelerated by a 2D AABB-tree.

We tag the extremities of each intersected edge as either *inside* or *outside* $\mathrm{P}_2(\mathcal{B}_1)$ according to their relative positions with respect to the quad. The closed region $\mathrm{P}_2(\mathcal{B}_1)$ is then obtained by propagating the *inside* tags using, once again, a flood fill algorithm. Eventually, the final mapping region $\mathcal{M}_2$ is obtained as the union of $\mathcal{B}_2$ and $\mathrm{P}_2(\mathcal{B}_1)$; this is required in more complex configurations than the one depicted in Figure 4.4 for which $\mathcal{B}_2$ is already included in $\mathrm{P}_2(\mathcal{B}_1)$.

Once the mapping regions have been determined for both meshes, we project all the vertices inside them by casting rays along $\hat{\mathbf{d}}$ (resp. $-\hat{\mathbf{d}}$) and finding the most distant point on the opposite mesh, which is now guaranteed to exist.

We can classify the exact boundary points of the mapping regions into three categories: (1) the intersection points resulting from the collision detection described in Section 1; (2) the intersection points of the projected contours with the opposite mesh, and (3)

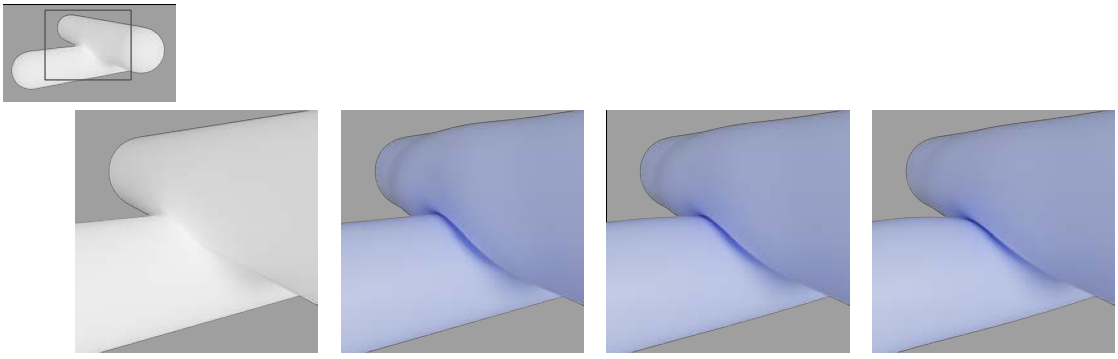


Figure 4.5.: Variations of the relative stiffness parameter $\bar{k}$ between two capsules. Here $\bar{k}$ is the relative stiffness of the lower capsule.

the contour points. For the first two categories, their mapping on the opposite mesh is already known by definition. For the third type of boundary points, we perform the same ray casting procedure as the one used for the vertices inside the mapping regions to find their associated point on the opposite mesh.

Discussion. Apart from the chaining with the boundary of $\mathcal{I}_l$ when necessary, the first step of this method is simply following [HZ00]. The challenging part is computing the intersection of the projected contours with the opposite mesh, especially when several contour segments intersect the same mesh edge. As in Section 1, we only keep track of the intersection points closest to each extremity of the current edge. Those are sufficient to determine the *inside* and *outside* tag of the edge vertices.

Note that, contrary to the previous approach (Section 2.1), the resulting mapping is not a bijection between two subsets of the initial surfaces. Indeed, for non-convex objects, rays along $\hat{\mathbf{d}}$ might intersect the opposite surface multiple times. However, we still obtain an injective map by retaining the furthest intersection as unique match. We will see in the following sections the other consequences of this lack of bijection.

3. Potential contact surface

Once the mapping regions have been defined for each object, we need to determine the shape of the shared contact surface $\mathcal{S}$ resulting from the collision. As its name suggests, the potential contact surface corresponds to a virtual surface located between each pairs of mapping regions. It describes the shape of the final interface between the two colliding objects as if they were in contact over their whole mapping regions.

In the limit asymmetric case involving a rigid object, the contact surface is the rigid mesh. For two elastic objects, this is not as straightforward; the intermediate surface is located between the two initial meshes and depends on their *relative stiffness*, defined

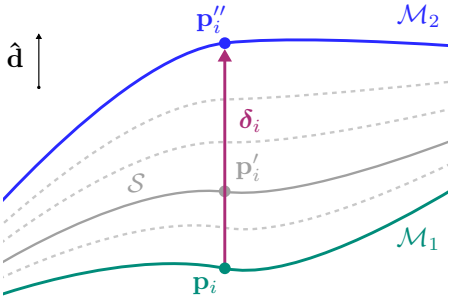


Figure 4.6.: Potential contact surface definition.

for mesh l as:

$$\bar{k}_l = \frac{k_l}{k_1 + k_2}, l \in \{1, 2\},$$

where $k_{l\in\{1,2\}}$ are user-controlled pseudo-stiffness parameters. The effect of $\bar{k}$ is depicted in Figure 4.5. The full projection of one mapping region onto the other, as defined in the previous section, corresponds to the displacement of a fully elastic surface onto a rigid object, i.e., a zero relative stiffness. Conversely, if the mapping region is not modified, it corresponds to a relative stiffness of 1. For intermediate values, we use a fraction of this displacement proportional to the relative stiffness of the considered mesh. For example, if a point $\mathbf{p}_i$ of $\mathcal{M}_1$ is mapped to $\mathbf{p}_i''$ on $\mathcal{M}_2$ by a displacement vector $\boldsymbol{\delta}_i = \mathbf{p}_i'' - \mathbf{p}_i$, its position on $\mathcal{S}$ will be $\mathbf{p}_i' = \mathbf{p}_i + (1 - \bar{k}_1)\boldsymbol{\delta}_i$

To determine the extent of the contact zone and ensure C^1 continuity at its boundary (as detailed in Section 4), we need the normal at each of these projected points. Independently projecting the polygonal surface of $\mathcal{M}_1$ and $\mathcal{M}_2$ onto $\mathcal{S}$ and recomputing vertex normals would require to explicitly cut each mesh along their mapping regions boundary, and more importantly, it would lead to discretization errors that would break the symmetry of the contact zone extraction as it is based on the normal field of the contact surface (see next section). Instead, for every vertex i of $\mathcal{M}_1$, we compute its normal on the shared potential surface $\mathcal{S}$ directly from its initial normal $\mathbf{n}_i$ and the interpolated normal $\mathbf{n}_i''$ of its corresponding point on $\mathcal{M}_2$, when $\mathcal{S}$ is defined using the above linear interpolation (see Figure 4.7).

Indeed, for the sake of symmetry, to compute the normal of each vertex projected on $\mathcal{S}$, we use an implicit surface representation of $\mathcal{S}$, and derive its normals by calculating the gradient of the associated scalar field at this position. Let us consider a point $\mathbf{x}$. $\mathrm{P}_1(\mathbf{x})$ and $\mathrm{P}_2(\mathbf{x})$ are its associated projection points along $\hat{\mathbf{d}}$ on $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. We want to find the gradient on $\mathcal{S}$ at $\mathbf{x}$. By definition, this point is at the intersection of $\mathcal{S}$ and the segment joining $\mathrm{P}_1(\mathbf{x})$ and $\mathrm{P}_2(\mathbf{x})$. We thus define the following scalar field:

$$F(\mathbf{x}) = ((\bar{k}_2\ \mathrm{P}_1(\mathbf{x}) + (1 - \bar{k}_2)\ \mathrm{P}_2(\mathbf{x})) - \mathbf{x})^\top \cdot \hat{\mathbf{d}}$$

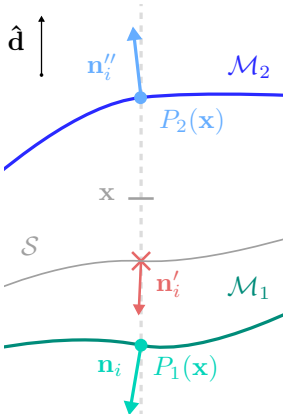


Figure 4.7.: Normals on $\mathcal{S}$.

The zero iso-surface $F = 0$ defines the implicit surface corresponding to $\mathcal{S}$. To obtain the normal at each point of this implicit surface, we now need to compute the gradient of this function:

$$\nabla F = \bar{k}_2\ \nabla(\mathrm{P}_1(\mathbf{x})^\top \cdot \hat{\mathbf{d}}) + (1 - \bar{k}_2)\ \nabla(\mathrm{P}_2(\mathbf{x})^\top \cdot \hat{\mathbf{d}}) - \hat{\mathbf{d}}.$$

Taking the linear approximation of P_1 and P_2, we finally get:

$$\nabla F = (I - \hat{\mathbf{d}}^\top \hat{\mathbf{d}}) \left(\frac{\bar{k}_2}{\mathbf{n}_1^\top \hat{\mathbf{d}}}\ \mathbf{n}_1 + \frac{(1 - \bar{k}_2)}{\mathbf{n}_2^\top \hat{\mathbf{d}}}\ \mathbf{n}_2 \right) - \hat{\mathbf{d}},$$

with $\mathbf{n}_1$ the normal on $\mathcal{M}_1$ at this point, and $\mathbf{n}_2$ the interpolated normal on $\mathcal{M}_2$ at $\mathrm{P}_2(\mathbf{x})$. To avoid numerical instabilities when normalizing the vector, we eliminate the denominators in the previous equation to obtain the final direction associated to $\mathcal{M}_1$:

$$\zeta_{i_1} = (I - \hat{\mathbf{d}}^\top \hat{\mathbf{d}}) \left(\frac{\bar{k}_2 \ |\mathbf{n}_{i_2}^\top \hat{\mathbf{d}}|}{\mathrm{sign}(\mathbf{n}_{i_1}^\top \hat{\mathbf{d}})} \ \mathbf{n}_{i_1} + \frac{(1 - \bar{k}_2) \ |\mathbf{n}_{i_1}^\top \hat{\mathbf{d}}|}{\mathrm{sign}(\mathbf{n}_{i_2}^\top \hat{\mathbf{d}})} \ \mathbf{n}_{i_2} \right) - |\mathbf{n}_{i_1}^\top \hat{\mathbf{d}}| \ |\mathbf{n}_{i_2}^\top \hat{\mathbf{d}}| \ \hat{\mathbf{d}},$$

The final normal of vertex i of $\mathcal{M}_1$ on $\mathcal{S}$ is then:

$$\mathbf{n}'_{i_1} = \frac{\zeta_{i_1}}{\|\zeta_{i_1}\|}.$$

To obtain the final normal of a vertex j of $\mathcal{M}_2$ on $\mathcal{S}$, we replace $\hat{\mathbf{d}}$ by $-\hat{\mathbf{d}}$ in previous equations.

Discussion The potential contact surface $\mathcal{S}$ is obtained by linear interpolation for simplicity and performance reasons. As a downside, due to the non-bijective definition of the mapping regions, $\mathcal{S}$ can exhibit C^0 discontinuities. One approach to this issue would be to extend the linear interpolation with continuity constraints over whole potential contact surface. Another solution could be to anticipate this problem earlier in the pipeline by directly modifying the definition of the mapping regions, as discussed in more details at the end of this chapter.

4. Contact definition

Eventually, we want to identify which part of each object will remain in contact with the opposite one after the deformation, that is which subset of the potential contact surface $\mathcal{S}$ will be considered inside the contact zone $\mathcal{C}$. We observed empirically that the extent of the contact zone between two objects in collision varies according to their respective geometry, stiffness and the depth of interpenetration. For example, the deeper the collision is, the wider the contact zone should be. Conversely, the stiffer the object is, the smaller this region will be. We will now present two methods to determine these contact zones considering these observations. The first one, detailed in Section 4.1, is limited to rigid-elastic contacts. It uses a volumetric criterion reflecting the pseudo-stiffness of the elastic object and taking into account the local geometry of $\mathcal{S}$ and the initial surface. The second approach, described in Section 4.2, is based on a surfacic geometrical symmetric criterion which allows to handle elastic-elastic collisions.

To comply with one of our main requirements, the criterion used by both approaches can be controlled by the user to tweak the apparent stiffness of the elastic objects involved in the collision. Moreover, as mentioned in Chapter 3, the deformable region extends from the contact zones boundary. Since we need a continuous definition of the deformation, the boundary of the contact region must evolve smoothly over the course of the animation.

4.1. Ball testing

Our first method only applies to the asymmetric case of a collision between a rigid and an elastic object. In such a case, $\mathcal{S}$ corresponds to the surface of the rigid object.

Inspired by α-shapes [EM94], our idea is to translate the stiffness of the elastic object into the radius r_s of a virtual ball that would roll on the interior of the elastic mesh after being projected on the rigid mesh (Figure 4.8 (a)). The contact zone corresponds to the sub-region of the surface accessible by this ball. The larger the ball, the least it can access concavities of the elastic surface, the smaller the contact zone will be. The contact zone will naturally expand as the depth of the intersection grows.

We developed a fast approximation of this test called *ball testing*. For each vertex with index i of the deformable region in intersection, we check whether any other deformable vertex is inside the ball of radius r_s and center $\mathbf{c}_i = \mathbf{p}'_i + r_s\mathbf{n}'_i$. If so, this vertex is not part of the contact zone (Figure 4.8 (b)). This test is accelerated using a 3D AABB-tree, built here over all the projected vertex positions.

However, defining the contact zone at vertices will not allow a smooth temporal evolution of its boundary. To refine it further, we compute the exact contact-boundary points on the outgoing edges of the contact zone by "sliding" the same virtual ball along them until it touches the elastic surface (Figure 4.9). More precisely, for each edge ij connecting a vertex with index i inside the contact zone to a vertex with index j outside this zone, the ball is defined by the constant radius r_s and the linearly interpolated center: $\mathbf{c}(\alpha) = (1-\alpha)\,\mathbf{c}_i + \alpha\,\mathbf{c}_j$, with $\alpha \in [0, 1]$ the parametric location along the edge. The exact contact-boundary point corresponds to the smallest α at which the ball is tangent to another vertex $\mathbf{p}'$ of the elastic mesh. This boils down to solving the quadratic equation:

$$\|\mathbf{p}' - \mathbf{c}(\alpha)\|^2 - r_s^2 = 0.$$

In practice, we perform this test on every vertex contained in the axis-aligned bounding box of the two spheres associated to the extremities of the edge (i.e., with center $\mathbf{c}(0)$ and $\mathbf{c}(1)$), leveraging the same 3D AABB-tree. If the edge contains intersection points (Section 1), then we clamp α as required to ensure that the contact-zone stays within

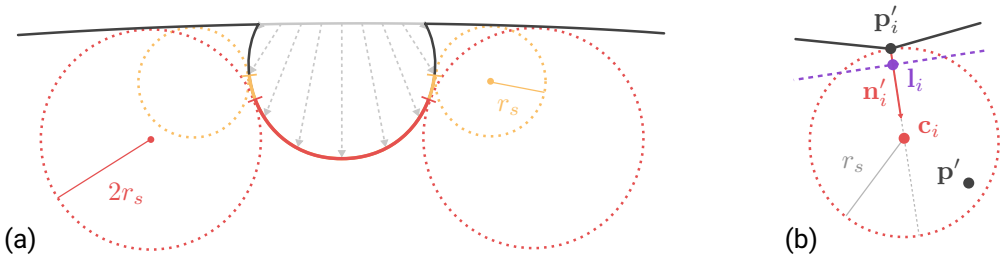


Figure 4.8.: Ball testing in 2D. (a) Contact zones (red and yellow solid lines) are identified on the projected elastic surface (dark gray solid line) by testing whether it can fit a ball of a given radius (r_s for the region in yellow, $2r_s$ for the one in red). (b) For a vertex at position $\mathbf{p}'_i$ with normal $\mathbf{n}'_i$, we test whether any other vertex $\mathbf{p}'$ of the elastic mesh is inside the ball with center $\mathbf{c}_i$ and radius r_s, excluding all vertices behind the hyperplane with normal $\mathbf{n}'_i$ passing through the point $\mathbf{l}_i$.

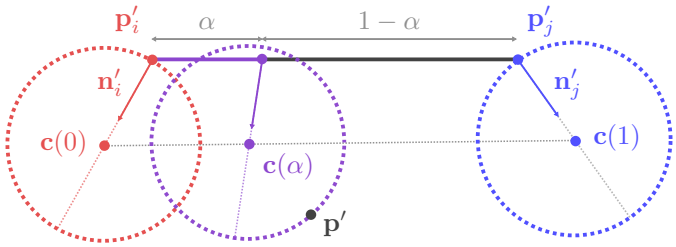


Figure 4.9.: Ball sliding in 2D. The contact zones extend across edges: for a vertex at position $\mathbf{p}'_i$ with normal $\mathbf{n}'_i$ inside the contact zone (in red) connected to a vertex with position $\mathbf{p}'_j$ with normal $\mathbf{n}'_j$ outside this zone (in blue), we search for the smallest parametric location α along this edge such that the associated ball (in purple) with center $\mathbf{c}(\alpha)$ and radius r_s is tangent to another vertex $\mathbf{p}'$ of the elastic mesh (the ball radius is usually much larger than the edge in practical situations).

the intersection. The overall smallest α is stored on the ij edge and denoted later as α_{ij}.

As we linearly interpolate the ball centers, the ball can slightly penetrate the deformed surface along the edge as seen in Figure 4.9, thus leading to false detection. Enforcing the ball to remain tangent to the edge would make the problem of finding the optimal α intractable. Moreover, since the points $\mathbf{p}'$ come from the projection onto a discrete surface, and not a continuous one, the above tests are sensitive to slight variations of the projection. We address both problems by excluding from ball testing all vertices lying behind an hyperplane of normal $\mathbf{n}'_i$ and passing through the point $\mathbf{l}_i = \mathbf{p}'_i + \lambda_i \mathbf{n}'_i$ (Figure 4.8 (b)). Here, λ_i is chosen as the smallest positive scalar such that the one-ring neighborhood of the current vertex lies in the rejection side of the hyperplane. This effectively excludes the topological one-ring-neighborhood while being compatible with the continuous ball-sliding test: the interpolated hyperplanes are obtained by linear interpolation of their respective normals $\mathbf{n}'_i$, $\mathbf{n}'_j$ and reference points $\mathbf{l}_i$, $\mathbf{l}_j$.

At the end of this step, the contact zone can be composed of multiple non-connected regions that can smoothly split and merge during the animation. Note that we never need to insert the contact points into the meshes, nor do we have to chain them together, which significantly simplifies computation. Figure 4.10 illustrates the effect of the variation of the pseudo-stiffness parameter r_s on the final deformation.

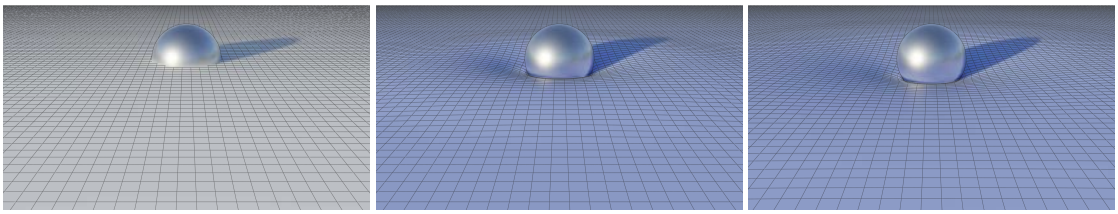


Figure 4.10.: Variations of the pseudo-stiffness parameter r_s of an elastic plane colliding with a rigid sphere. This parameter controls the extent of the contact zone. To produce plausible results, the extent of the deformation area has been increased accordingly.

Discussion. This approach has undeniable advantages: it is continuous and produces coherent contact regions that evolve in a plausible fashion. However, it is inherently limited to rigid-elastic collisions due to its asymmetric definition. Indeed, for elastic-elastic contacts, the contact zones $\mathcal{C}_l$ of each mesh l should result in the same projected regions on $\mathcal{S}$ since the contact is shared by the two objects. The issue does not come from $\mathcal{S}$, as its geometry is the same for both meshes, but from the ball testing procedure itself. By construction, it computes the final contact zones taking into account parts of the mesh geometry that can be outside the collision, i.e., outside $\mathcal{S}$. Therefore, it is highly unlikely to find adequate testing spheres for both meshes to obtain consistent contact zones.

In addition, we observed that the stiffer an object is, the less its initial curvature can change. Consider for instance a bar with zero curvature, such as a ruler: if it is very rigid, a high force must be applied to bend it; the more elastic the ruler becomes, the weaker the force required to bend it is. At equal force magnitude, the bar can reach a higher curvature. The same idea applies to a ball. Consider a very rigid bowling ball, it is very unlikely that we can change its curvature by pressing on it. Conversely, if the ball is a stress ball, we can squeeze it more easily and thus increase its curvature around the contact. Using our method and changing user parameters to achieve the desired deformation on different objects allow to reproduce these observations. It also leads us to the idea that the radius of the testing sphere should not be the same over the whole mesh, but should rather depend on the initial local curvature of the considered object. We did not pursue this idea further, but it would be interesting, in future work, to study the effect of this spatially-varying radius.

Moreover, we made the observation that the thickness of the object should be taken into account. Indeed, the thickness of an object has an impact on its stiffness. In addition, with our volumetric criterion, we assume that, before collision, the testing ball is able to *roll* over the inside of the initial surface without any collision with the rest of the surface. For objects with varying thicknesses, our intuition is that the maximum radius of the ball should change depending on the *local feature size*, that is, the distance to the medial axis. Such a radius could be precomputed as a preprocess and acts as a upper bound for the radius set by the user. We did not put efforts in this direction yet as this question is also connected to the application domain of our method. Indeed, scenarii for which adapting the ball radius would make sense would very likely also require a *global* deformation response (e.g., through skinning), that our local deformer cannot achieve anyway.

4.2. Geometrical method

To generalize the definition of the contact zone to the case of two elastic objects, an additional constraint is required: *symmetry*. As the contact is defined on the potential contact surface $\mathcal{S}$, the contact zones on both meshes should correspond to the same regions on this common intermediate surface (i.e., result in the same projected zones along $\hat{\mathbf{d}}$ (resp. $-\hat{\mathbf{d}}$) on $\mathcal{S}$).

We thus developed a second method based on a criterion for belonging to a contact

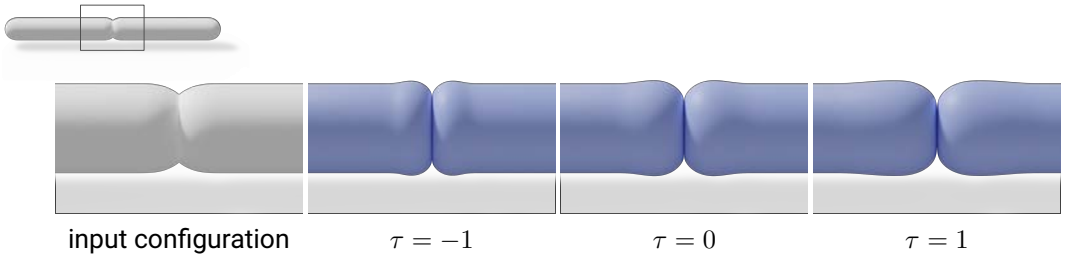


Figure 4.11.: Variations of the apparent stiffness parameter τ between two capsules. This parameter controls the extent of the contact zone. To produce plausible results, the extent of the deformation area has been increased accordingly.

zone that captures all variations of geometry, stiffness and depth while satisfying our application requirements, i.e., temporal continuity, artistic controllability, and symmetry with respect to the two objects.

To obtain a symmetric behavior, this criterion is based on the geometry of the common surface $\mathcal{S}$, and motivated by two empirical observations. On the one hand, the more distant the normal of this surface is from the direction of collision, the less the two objects should remain in contact. On the other hand, the deeper the collision is, the larger the contact zone should be. Therefore, we define our criterion as follows. On each mapping region $\mathcal{M}_l$, a vertex remains on the contact surface $\mathcal{S}$, that is, in the contact zone $\mathcal{C}_l$, if it fulfills the condition:

$$\boldsymbol{\delta}_i^\top \mathbf{n}'_i \geq \varepsilon_c, \tag{4.2}$$

where, as before, $\mathbf{n}'_i$ is the normal of vertex i on $\mathcal{S}$, and $\boldsymbol{\delta}_i$ is the displacement between the vertex position and its projection on the other mesh. Since the average magnitude of those scalar products largely varies over the animation, the threshold ε_c needs to be continuously recomputed. To ensure a stable control during the animation, our idea is to start from the mean $m_\mathcal{S}$ of $\boldsymbol{\delta}_i^\top \mathbf{n}'_i$ over $\mathcal{S}$, and then deviate from this reference by a user-controlled amount τ of the standard deviation $\sigma_\mathcal{S}$ of $\boldsymbol{\delta}_i^\top \mathbf{n}'_i$ over $\mathcal{S}$:

$$\varepsilon_c = m_\mathcal{S} + \tau\sigma_\mathcal{S}. \tag{4.3}$$

To ensure temporal continuity, $m_\mathcal{S}$ and $\sigma_\mathcal{S}$ are computed by integrating $\boldsymbol{\delta}^\top \mathbf{n}'$ over $\mathcal{M}_1$ and $\mathcal{M}_2$ using Appendix A; The integration is limited by the exact boundary points of the mapping region defined on mesh edges, similarly to $\bar{\mathbf{n}}_l$ and $\mathrm{Var}(\mathbf{n}_l)$ on $\mathcal{W}_l$ in Section 2.2. As depicted in Figure 4.11, the scale-independent parameter τ allows the user to adjust the apparent stiffness of the most elastic object by controlling the extent of the contact zone, while its shape is determined by the relative stiffness $\bar{k}_l$ defining $\mathcal{S}$.

Moreover, to avoid too high tangential displacements in the final deformation, we add to Equation 4.2 the following safeguard symmetric condition on the angle between the initial surface normals and the mapping direction:

$$\min(-\hat{\mathbf{d}}^\top \mathbf{n}_i, \hat{\mathbf{d}}^\top \mathbf{n}''_i) \geq \cos(80^\circ),$$

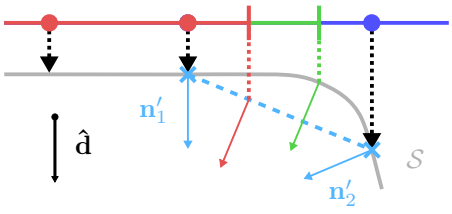
where $\mathbf{n}''_i$ denotes as before the normal of the point corresponding to the vertex i on the opposite object (see Figure 4.6).

With such a definition, the contact zone is discretized at the vertices of both meshes. Once more, we need a smooth temporal evolution of its boundary to guarantee a continuous deformation during animation, and we thus compute the exact-boundary points of the contact zone on mesh edges. For each outgoing edge ij joining a vertex i inside the contact zone and a vertex j outside of it, we find the smallest barycentric coordinate α_{ij} at which the following conditions are met:

$$\left.\begin{aligned} \mathbf{n}'(\alpha)^\top \boldsymbol{\delta}(\alpha) &\geq \varepsilon_c \\ \min(-\mathbf{n}(\alpha)_1^\top \hat{\mathbf{d}}, \mathbf{n}(\alpha)_2^\top \hat{\mathbf{d}}) &\geq cos(80°) \end{aligned}\right\} \tag{4.4}$$

where $\mathbf{n}'(\alpha)$ is the normal of that point on $\mathcal{S}$, $\boldsymbol{\delta}(\alpha)$ is its displacement, and $\mathbf{n}(\alpha)_{l\in\{1,2\}}$ are its initial normals on each mesh; all those quantities are independently obtained by linear interpolation of their respective values at the edge extremities. Once the minimum α value has be obtained by the resolution of a second degree equation, we project the corresponding interpolated point onto $\mathcal{S}$ following $\hat{\mathbf{d}}$ (or $-\hat{\mathbf{d}}$) and recompute its exact final position and normal that will be used in the subsequent steps of the pipeline. These exact-boundary points are stored using their barycentric coordinates, denoted later as α_{ij}, on their associated edge ij along with their corresponding projected point and normal on $\mathcal{S}$.

Discussion. Although this method has the major advantage of handling elastic-elastic contacts, some of the strengths of the ball testing approach are lost. In this second method, ε_c is computed and used for the whole mesh, hence this criterion cannot adapt to local variations of the shapes. By comparison, even with a constant sphere radius, the first method solely takes into account the local geometry of the mesh to assess whether each tested vertex belongs to the contact zone; the ball-testing criterion is thus much more local. This can have a significant impact on certain configurations. Consider for instance a slightly elastic palm pressing onto a soft cushion, we expect the collision of each finger to be treated independently, or at most by considering the directly adjacent fingers. The collision depth of the forefinger should have little, if any, influence on the contact of the little finger. This will not be the case with our global surfacic criterion ε_c, which can lead to missed contact zones giving the impression of missed collisions. As future work, it would be interesting to find a method combining the locality of the ball testing approach and symmetric definition of our second criterion.



In this second approach, we also made some approximations leading to potential issues with high curvature objects. To determine exact boundary contact points, we use a degree two equation linearizing $\mathbf{n}'$, the normal on $\mathcal{S}$, and $\boldsymbol{\delta}$, the displacement between the vertex position and its projection on the other mesh. We thus make the assumption that the distance to the opposite mesh varies linearly along the considered edge. For meshes with low curvature this approximation is acceptable, but as the curvature increases, as shown in the inset figure, this can lead to an imprecise definition of the

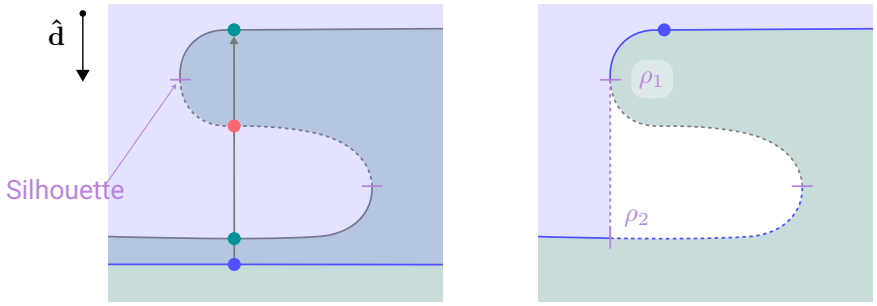
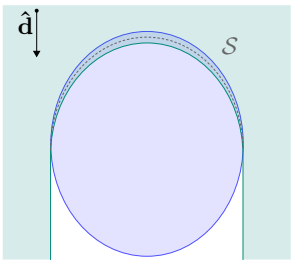


Figure 4.12.: Issue with concavities. A plane (in blue), coming from the top, collides with a surface with a fold (in green). Right: due to the fold, the blue point of the plane can be mapped to two green points of the mapping region of the green surface: our mapping is thus not bijective. Left: the solution would be to remove the parts of the mapping region (here at the bottom) that can be projected on the same object following $\hat{\mathbf{d}}$. The top contour point ρ_1 will be projected on ρ_2. From the blue surface perspective, the corresponding point should be split into two to get the displacement magnitude and normal on $\mathcal{S}$ from both points on the green surface.

boundary of the contact zone. In red we see the point determined by our method and in green the real boundary point. A solution could be, once a first approximation of the exact contact point barycentric coordinate α is found by the method described above, to verify the normal $\mathbf{n}'(\alpha)$ and displacement $\boldsymbol{\delta}(\alpha)$ with its projection on the opposite mesh. If the dot product between those two quantities are too far from the computed criterion ε_c, we could refine the position of the boundary point along the current edge by dichotomy, assuming that the dot product is monotonic, which should be the case for low frequency surfaces.

Another welcome property of the ball testing method that we have lost with this second approach is illustrated in the inset figure. In the special case of the collision between a concave surface and a convex surface with opposite curvature, such as a saddle on the back of a horse, we expect the contact region to extend over the whole intersection region. However, if both objects are quite rigid, due to our choice of a unique mapping direction, we will not get the expected result. The contact will be mostly located on the top of the green ellipsoid shape due to the alignment of the normal on $\mathcal{S}$ with $\hat{\mathbf{d}}$. The ball testing approach would produce a much better result: the entire intersection region would be part of the contact zone.



Finally, the last topic that we want to discuss is the case of concavities as illustrated in Figure 4.12. Here we consider the blue plane as elastic and the green surface with the concavity as rigid. As already mentioned, the mapping presented in Section 2.2 is only injective in such configurations. It can thus exhibit discontinuities that can lead to some issues. First, in Figure 4.12, we can see that a subset of the green mapping region (solid line in the left sub-figure) is not *reachable* by the blue one (dashed blue line in the right sub-figure). Yet, this portion of the surface is part of $\mathcal{S}$ in our definition and thus contributes to the computation of ε_c, which is then biased. This subset should thus be removed from the mapping region of the green surface, because it corresponds to only

one of the two objects.

The question is now how to obtain a plausible contact region with such a discontinuity. To avoid *transferring* the discontinuity of $\mathcal{S}$ to $\mathcal{C}$, the contact boundary should move away from ρ_2 to its left. With the same idea of curvature variation mentioned previously, the blue surface cannot closely fit this discontinuity to create a sharp angle unless this object is very elastic. Our surfacic criterion does not seem suitable in such configuration because, if one vertex of the blue mesh projected on the lower part of the green surface is detected as part of the contact zone, there is no reason for this region to extend up to ρ_2, and the discontinuity of $\mathcal{S}$ will be propagated to $\mathcal{C}$. Again, our volumetric ball-testing method will seamlessly handle such a configuration provided a large enough ball radius.

One possibility to address all the aforementioned curvature-related issues would be to take inspiration from the surfacic opening method recently proposed by Sellán et al. [SKSJ20]. Contrary to the ball testing, this is a surfacic method, but it is based on the local curvature of the surface. The part of mesh that stays put on $\mathcal{S}$ at the end of the process can be considered in the contact region. The drawbacks of this method are the required remeshing and the iterative process to define the output shape. Similarly to the ball testing, we also need to choose an appropriate size of the structuring element. However this idea is a promising research direction to find a good compromise between our volumetric and surfacic methods.

5. Algorithm

In this section, we summarize with an algorithm the different steps described in this chapter with all the possible alternatives.

Our method can take as input more than two objects. When using the mapping version described in Section 2.1, each object can represent a working region. But, as mentioned earlier, when using a unique mapping direction for the matching step (Section 2.2), the direction $\hat{\mathbf{d}}$ is related to the instantaneous movement of the considered objects. According to its movement each object can be processed as either a unique working region if its displacement is global (i.e., the same for each part of the input object), or multiple disconnected components if they follow different directions. In that case, they must be considered as several independent working regions that will be processed separately.

In general, we consider a set of working regions $\{\mathcal{W}_l\}$, represented by triangle meshes, that may belong to a single or multiple objects. The first steps of our pipeline to extract the contact zones are performed on each pair of working regions ($\mathcal{W}_a$, $\mathcal{W}_b$) in collision. At the end of these steps, after the extraction of the contact zone on each surface, we store for each working region l the sets $\mathcal{J}_l$ of working regions with which they are in

The following algorithm corresponds, on the left, to the pipeline described in [BBG21], illustrated on the right part of Figure 3.1 (steps (a) to (d)), and the blocks on the right are the alternatives used in [BBGB20], corresponding to the left part of the same figure.

Algorithm Part 1 - Contact

Input: a set of triangle mesh working regions $\{\mathcal{W}_l\}$
Output: the deformed elastic surface positions $\mathbf{p}'$

for all pairs $(\mathcal{W}_a, \mathcal{W}_b)$ in intersection **do**

Intersection detection ▷ §1
Compute and store edge-face intersection points,
yielding intersection regions $\mathcal{I}_a$, $\mathcal{I}_b$

Mapping direction ▷ § 2.2
If not provided, estimate $\hat{\mathbf{d}}_{a,b}$ through integrals
over $\mathcal{I}_a$ and $\mathcal{I}_b$ using Eq. 5.1 ($\hat{\mathbf{d}}_{b,a} \leftarrow -\hat{\mathbf{d}}_{a,b}$)

Region mapping
for all $(a', b') \in \{(a,b),(b,a)\}$ **do**
$Q \leftarrow \{\}$ ▷ set of extruded contour quads
for all face f in $\mathcal{I}_{a'}$ exhibiting a contour segment s **do**
$Q \leftarrow Q \cup \{\text{extruded\_quad}(s, \hat{\mathbf{d}}_{a',b'})\}$
for all front facing edge ij of $\mathcal{W}_{b'}$ intersecting Q **do**
Tag its extremities as either *inside* or *outside*
Propagate the *inside* tag yielding $\mathrm{P}_{b'}(\mathcal{B}_{a'})$
$\mathcal{M}_{b'} \leftarrow \mathrm{P}_{b'}(\mathcal{B}_{a'}) \cup \mathcal{B}_{b'}$
for all vertex $\mathbf{p}_i$ in $\mathcal{M}_{b'}$ **do**
$\mathbf{p}''_i \leftarrow \text{farthest\_intersection}(\mathcal{W}_{a'}, \text{ray}(\mathbf{p}_i, \hat{\mathbf{d}}_{a',b'}))$

Region mapping ▷ §2.1
Find affine transform A minimizing $\sum_k \|\mathbf{e}_k\|^2$,
with $\mathbf{e}_k = \mathbf{q}_{k,a} - \mathrm{A}\,\mathbf{q}_{k,b}$
Compute deformation field $\mathbf{g}$ by harmonic diffusion of $\mathbf{e}_k$
for all vertex $i \in \mathcal{W}_a \cap \mathcal{W}_b$ **do**
$\mathbf{p}''_i \leftarrow$ 3D position of $\mathrm{A}\,\mathbf{q}_{i,b} + \mathbf{g}_i$ on the rigid mesh

Potential contact surface $\mathcal{S}$ ▷ §3
for all $(a', b') \in \{(a,b),(b,a)\}$ **do**
for all vertex i in $\mathcal{M}_{a'}$ **do**
$\mathbf{p}'_i \leftarrow \mathbf{p}_i + (1 - \frac{k'_a}{k_a + k_b})(\mathbf{p}''_i - \mathbf{p}_i)$

Contact zone extraction ▷ §4.2
Compute the threshold ε_c using Eq. 4.3 and integrals over $\mathcal{S}$
for all $l \in \{a, b\}$ **do**
$\mathcal{C}_l \leftarrow \{\mathbf{p}_i,\ i \in \mathcal{M}_l \mid \mathbf{p}_i \text{ satisfies Eq. 4.2}\}$
for all edge ij with $i \in \mathcal{C}_l$, and $j \notin \mathcal{C}_l$ **do**
Find α_{ij} using Eq. 4.4
$\partial\mathcal{C}_l \leftarrow \partial\mathcal{C}_l \cup \{(1 - \alpha_{ij})\,\mathbf{p}_i + \alpha_{ij}\,\mathbf{p}_j\}$

Contact zone extraction ▷ §4.1
for all $i \in \mathcal{M}_b$ **do** ▷ Ball testing, supposing $\mathcal{W}_a$ rigid
if no $\mathbf{p}'_{j \neq i}$ inside the ball with radius r_s
and center $\mathbf{c}_i = \mathbf{p}'_i + r_s \mathbf{n}'_i$ **then**
$\mathcal{C}_b \leftarrow \mathcal{C}_b \cup \{\mathbf{p}'_i\}$
for all edge ij with $i \in \mathcal{C}_b$, and $j \notin \mathcal{C}_b$ **do** ▷ Ball sliding
Find smallest α such as $\|\mathbf{p}' - ((1-\alpha)\,\mathbf{c}_i + \alpha\,\mathbf{c}_j)\|^2 - r_s^2 = 0$
$\partial\mathcal{C}_b \leftarrow \partial\mathcal{C}_b \cup \{(1 - \alpha)\,\mathbf{p}'_i + \alpha\,\mathbf{p}'_j\}$

if $\mathcal{C}_a$ and $\mathcal{C}_b$ are not empty **then**
$\mathcal{J}_a \leftarrow \mathcal{J}_a \cup \{b\}$; $\mathcal{J}_b \leftarrow \mathcal{J}_b \cup \{a\}$ ▷ sets of contact zone indices

end ▷ continues in Part 2

5

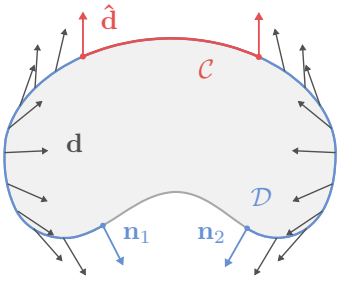
Deformation

In this chapter we present the second steps of our method corresponding to the last row of Figure 3.1. After the definition of the region where the colliding objects will stay in contact, we must determine the resulting deformation surrounding it. The objective is to compute the final position of the involved surfaces around their contact regions. In the previous chapter, we processed each pair of colliding working regions individually. In this chapter, however, the deformation of each working region is computed independently of the others. This means that we will focus on computing the deformation of a single working region at once, considering the single or multiple contact zones coming from the collision of this working region with one or multiple other working regions. In the later case, the working region includes multiple contact regions with potentially different mapping directions.

As mentioned in Chapter 3, several approaches could be conceived to compute the final deformation, and we made the key decision to separate the computation of the final displacement into two parts: its *direction* and its *magnitude*. Although this decision already narrows the range of possibilities, multiple approaches are still conceivable and we explored some of them. In the following, we describe those to explain the progression towards the final solution.

Motivation. First, a smooth direction field can be obtained by harmonic diffusion, interpolating, on the one hand, the mapping directions at the boundary of the contact zone and, on the other hand, the normals of the initial surface at the exterior boundary of the deformable region. This idea, that seems the most intuitive at first, does not produce the expected result on many cases.

For instance, let us consider the direction field on a bean-shaped elastic surface, as illustrated in the inset figure with a sectional drawing of the initial mesh. The contact region and the associated constrained mapping directions are colored in red and the deformable region $\mathcal{D}$ in light blue. $\mathbf{n}_1$ and $\mathbf{n}_2$ are the normals of the initial surface at the boundary of $\mathcal{D}$. In this configuration, the vector field $\mathbf{d}$ resulting



for harmonic diffusion of those 3D vectors does not point in the correct, or at least expected, direction. To avoid this unwanted behavior, we thus need to take into account the change of orientation over the surface, as described in Section 2.

Concerning the magnitude of the displacement, assuming we have a smooth direction field, we can obtain a smooth scalar field using poly-harmonic interpolation between $\partial\mathcal{D}$ and $\partial\mathcal{C}$. The volume preservation constraint can be applied as a soft constraint, and the resolution of our problem then boils down to an energy minimization. Using a simple Laplacian interpolation operator has the advantage of the performance but is not an option due to the tangential discontinuity at the boundaries of the considered region.

To improve the surface continuity on $\partial\mathcal{D}$ and $\partial\mathcal{C}$, the Laplacian operator can be replaced by a bi-Laplacian, at the cost of a biharmonic diffusion. Using a weighted bi-Laplacian instead of a uniform formulation, we can even reinforce the influence of the operator at the boundary of $\mathcal{D}$ over the volume preservation constraint. Yet, these improvements are still insufficient to robustly avoid discontinuities of the final deformed surface mainly due to the volume constraint. Moreover, it could be difficult to compute and tweak the weights to obtain the desired plausible deformation for each mesh and each collision configuration.

Finally, we can consider an ARAP energy formulation [SA07]. For performance reasons, it can be linearized [VGB+14] by fixing *a priori* the rotations to align the normals of the surface interpolated between the ones at boundary of $\mathcal{C}$ on $\mathcal{S}$ and the ones of the initial surface at the external boundary of $\mathcal{D}$. With this scheme, we get rid of discontinuities, but the bulge which is expected by the volume redistribution is flattened due to the nature of this energy. Indeed, the fixed rotations converge quickly to the normals of the initial surfaces, which makes the deformation closely follow the initial shape of the object, and therefore does not permit to produce a natural and plausible bulge. Moreover, to prevent the surface from stretching too much, the ARAP energy distributes the bulge more than expected, giving the impression of flatness. In extreme cases, coupled with the volume preservation constraint, it can even produce more discontinuities at the boundary of the deformable region.

In fact, we already know the kind of profile wanted for the deformation to reproduce the bulge. An idea would thus be to constrain the ARAP energy over this profile. According to our observation, an inflexion in the normal field of the final deformed surface is needed to obtain a plausible deformation. It cannot be obtained using a simple interpolation of the normals at the boundaries of $\mathcal{D}$ with the linearized ARAP formulation. It would be possible to design a target deformation using a profile curve and compute the corresponding rotation (needed by the chosen energy) to induce the inflexion on the surface.

However, in addition to the complexity of the process, this method suffers from the same issues as the previous ones: the indirection created by this profile curve controlling the rotation of the ARAP energy is difficult to adjust to obtain a plausible deformation. If we use a profile curve, we might as well control the displacement amplitude directly to remove this indirection.

Last but not least, all of these methods share a critical limitation: *the lack of artistic control*.

Our solution. To circumvent this limitations while complying with our initial requirements, we decided to compute the final position of the considered surfaces around their contact regions using a pure geometric displacement along a smooth direction field $\mathbf{d}$. As a reminder, the final position $\mathbf{p}'_i$ of every vertex $\mathbf{p_i}$ of $\mathcal{D}$ is obtained by the following equation:

$$\mathbf{p}'_i \quad = \mathbf{p}_i + \mathcal{H}_{a_i,s_i}(u_i)\,\mathbf{d}_i. \tag{5.1}$$

The displacement magnitude is controlled by a 1D profile function $\mathcal{H}$ defined over a 1D radial parametrization $u \in [0,1]$ of the deformable region. To ensure a smooth transition with the contact region, an adjusted profile $\mathcal{H}_{a_i,s_i}$ is instantiated at each vertex $\mathbf{p}_i$ of the deformable region $\mathcal{D}$ by fixing its amplitude a_i and slope s_i at $u = 0$. Since this information is only known at the contact-zone boundary, we have to diffuse it to the rest of the deformable region yielding the two corresponding smooth scalar fields a and s.

Note that s_i reflects the slope of the displacement in the direction orthogonal to the boundary of the contact region, which is thus unrelated to the gradient of a_i. To illustrate that point let us consider a simple symmetric configuration like a rigid sphere colliding an elastic plane. In this case, the amplitude a_i along the boundary of the contact zone is expected to be constant, and thus the field a will also be constant over $\mathcal{D}$. Its gradient will be a null vector field, whereas the scalar field s will also be constant over $\mathcal{D}$, because its value will reflect the slope of the sphere on the direction orthogonal to the boundary of the contact region.

To summarize, once the contact zones have been established for all pairs of colliding working regions, each of them is processed independently by computing the different ingredients of Equation 5.1:

1. the extent of the deformation together with a 1D parametrization of the corresponding deformable region (Section 1),
2. a directional field $\mathbf{d}$ supporting the vertex displacement (Section 2),
3. the guiding amplitude a and slope s fields (Section 3),
4. the final deformation of the surface along $\mathbf{d}$ (Section 4).

These four steps are described in the next sections.

1. Deformable region

The 1D parametrization of the deformable region, denoted u, must range from 0 at the contact region boundary $\partial\mathcal{C}$ to 1 at the exterior boundary of the deformable region. We define it as $u = \phi/\phi_{\max}$, where ϕ is a scalar field measuring a smoothed geodesic distance to the contact boundary $\partial\mathcal{C}$, and $\phi_{\max}$ is a user-specified maximum geodesic distance controlling the extent of the deformation (Figure 5.1). Formally, the exterior boundary of the deformable region is defined as the iso-contour $\phi = \phi_{\max}$.

Our computation of ϕ is inspired by the state-of-the-art heat-method [CWW13], which is a three-steps procedure:

1. Compute a scalar field v through heat diffusion.
2. Compute the normalized gradient field $X = -\nabla v/\|\nabla v\|$.
3. Solve the Poisson equation $\Delta\phi = \nabla \cdot X$.

Our approach mostly departs from the original method in the first step. Let us recall that at this stage we work on an open subset $\mathcal{W}$ of the initial surface. In such a case, Crane et al. recommend to compute v as the average of the two heat-diffusion solutions obtained with vanishing Neumann and Dirichlet conditions respectively, which we found too expensive for us. From our experience, Dirichlet conditions alone yield significantly better and stable solutions than natural Neumann conditions, especially if $\mathcal{W}_l$ is large enough. In practice, we use a time step equals to $10\,h^2$, with h the average edge length in the current working region.

Since we are seeking for a smooth distance field (implying a very large time-step in the heat-diffusion), we could consider simplifying the computation by using the solution of the steady state problem:

$$\Delta v = 0 \quad \text{with} \quad v = 1 \,\text{on}\, \partial\mathcal{C} \,\text{and}\, v = 0 \,\text{on}\, \partial\mathcal{W}$$

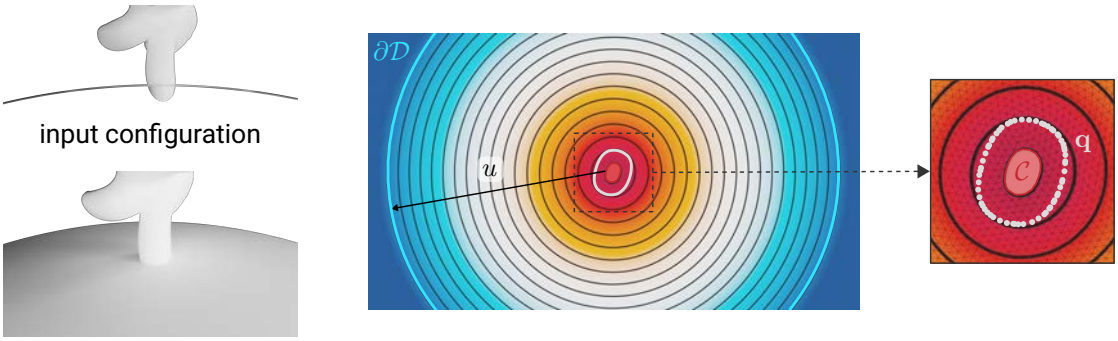


Figure 5.1.: Visualization of the parametrization on an elastic balloon pressed by a rigid finger. Center & right: Top view of the elastic surface, showing the boundary of the intersection region $\mathbf{q}$ (in white), the contact region $\mathcal{C}$, and the deformable region boundary $\partial\mathcal{D}$. The radial parametrization u (colored periodic iso-values) is 0 (red) on the contact boundary $\partial\mathcal{C}$ and 1 (blue) on $\partial\mathcal{D}$.

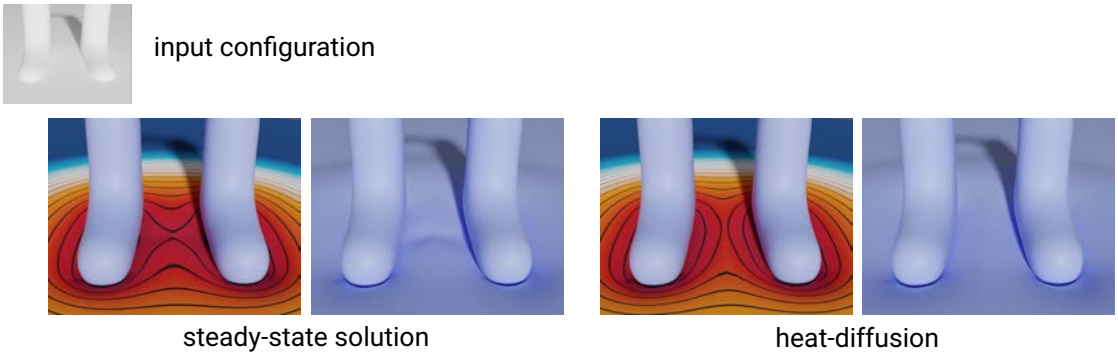


Figure 5.2.: 1D parametrization comparison. A more geodesic-like parametrization (right) avoids unpleasant distortions when multiple contact zones are involved in the collisions.

where $\partial\mathcal{W}$ denotes the exterior boundary of $\mathcal{W}$. However, even though the distance field produced by the steady-state solution is smoother, in practice it results in odd deformations as its isolines depart too much from the ones of the true geodesic distance field, as illustrated in Figure 5.2.

Since the contact boundary $\partial\mathcal{C}$ is not defined on existing vertices of the mesh but as an arbitrary polyline along its edges, we implement the Dirichlet boundary condition along $\partial\mathcal{C}$ as a set of least-square linear constraints $(1-\alpha_{ij})\,v_i + \alpha_{ij}\,v_j = 1$ for all edges ij crossed by the contact boundary. To avoid temporal discontinuities when the contact boundary $\partial\mathcal{C}$ crosses a vertex, we also use adequately weighted cotangent coefficients (see Appendix B).

When solving the Poisson equation in step 3, we reuse the same adjusted cotangent weights to compute the integrated divergences, and the same least-square constraints to ensure that $\phi = 0$ on $\partial\mathcal{C}$.

The result of this process is depicted in Figure 5.1. The parametrization respects the boundary constraints, is smooth, almost linear, and radially symmetric when expected.

2. Direction field

As a reminder, the direction field required by our method is subject to two constraints: it must match the fixed displacements of the contact zone at $u = 0$ and, for a smooth shape, it should be mostly aligned with the surface normals and eventually reached these normals at $u = 1$. The natural idea to compute such a direction field would be to use harmonic diffusion with Dirichlet constraints on $\partial\mathcal{C}$ and $\partial\mathcal{D}$. However, as presented in the introduction of this chapter, this method does not work as expected. We thus developed a method presented in Section 2.1 that alleviates this issue by coupling harmonic diffusion with parallel transport. However this process is costly due to the diffusion of complex values. Therefore, focusing on the speed of the transition towards the surface normals, we developed a much simpler method, described in Section 2.2.

2.1. Direction field diffusion using parallel transport

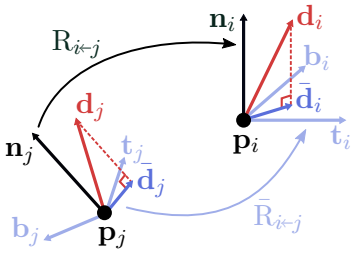
As introduced above, using a simple harmonic diffusion is not suitable for our goals. Indeed, even though boundary constraints are respected, the geometry of the initial surface between those boundaries is not considered. We thus developed a method that takes the variations of orientation of the undeformed surface into account.

For each vertex $\mathbf{p}_i$ of the exterior boundary of the deformable region, we impose $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i = \mathbf{n}_i$. Likewise, for each edge ij crossed by the contact boundary $\partial\mathcal{C}$, we should have:

$$\mathbf{d}(\mathbf{p}_{ij}) = \mathbf{d}_{ij} \ , \text{ with } \mathbf{d}_{ij} = \frac{\mathbf{p}_{ij} - \mathbf{p}'_{ij}}{\|\mathbf{p}_{ij} - \mathbf{p}'_{ij}\|} \, ,$$

where $\mathbf{p}_{ij} = (1-\alpha_{ij})\mathbf{p}_i + \alpha_{ij}\mathbf{p}_j$ is the exact boundary point along the edge, and $\mathbf{p}'_{ij}$ is its projection onto the opposite surface as obtained from the mapping defined in Chapter 4. To take into account the change of orientation over the surface, we encode each displacement direction $\mathbf{d}_i$ as its 2D projection onto its local tangent plane:

$$\bar{\mathbf{d}}_i = \mathrm{B}_i^\top \mathbf{d}_i \, ,$$



where the precomputed 3×2 matrix $\mathrm{B}_i = [\mathbf{t}_i \ \mathbf{b}_i]$ holds a pair of unit and orthogonal tangent vectors (inset figure). Intuitively, $\bar{\mathbf{d}}_i$ represents how much the direction deviates from its normal in tangent space. This also has the advantage of relaxing the non-linear unit vector constraint during the diffusion. However, we notice that, due to this projection, the diffused value would correspond to the sine of the angle between the initial normal $\mathbf{n}_i$ and the diffused unit direction $\mathbf{d}_i$, whose speed of variation is thus slower than the one of the final diffused angles, especially for large angles. For deep collisions, when the constraints at the two boundaries of the deformable region are significantly different, this method produces swiftly varying directions, resulting in unnatural deformations, and even folds, especially when the distance between the two boundaries is short. In such a configuration, the diffused direction varies actually too quickly to the surface normals. To counterbalance this effect, we apply the arcsin function to the constraints at the boundary of the contact zone before the diffusion:

$$\bar{\mathbf{d}}_{i(\text{angle})} = \arcsin(\bar{\mathbf{d}}_i) \, \frac{\bar{\mathbf{d}}_i}{\|\bar{\mathbf{d}}_i\|} .$$

A direction $\mathbf{d}_i$ is then recovered as:

$$\mathbf{d}_i = \mathrm{B}_i \, \bar{\mathbf{d}}_i + \sqrt{1 - \|\bar{\mathbf{d}}_i\|^2} \, \mathbf{n}_i \, , \tag{5.2}$$

$$\text{with } \ \bar{\mathbf{d}}_i = \sin(\|\bar{\mathbf{d}}_{i(\text{angle})}\|) \, \frac{\bar{\mathbf{d}}_{i(\text{angle})}}{\|\bar{\mathbf{d}}_{i(\text{angle})}\|} \tag{5.3}$$

This reduction of dimensionality is possible because we assume that the fixed directions at the contact boundary are directed towards the exterior of the deformable surface, which is a reasonable assumption for smooth surfaces.

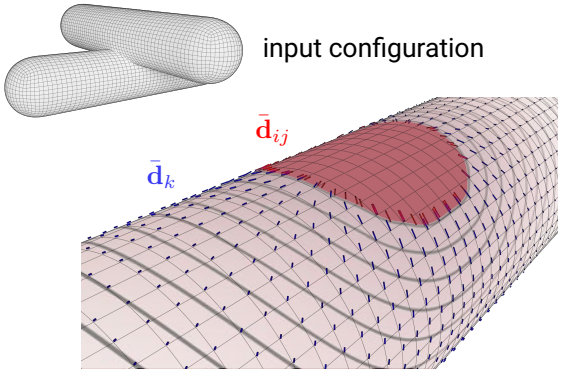


Figure 5.3.: Visualization of the direction field. The 2D projection of the direction field onto the surface tangent plane $\bar{\mathbf{d}}$ (in blue) is diffused from the value constraints $\bar{\mathbf{d}}_{ij}$ (in red). Here the constrained directions on $\partial\mathcal{C}$ are obtained using the mapping method presented in Section 2.1 of Chapter 4.

We interpolate these tangent vectors between the two boundaries using harmonic diffusion and parallel transport. Since two adjacent vectors $\bar{\mathbf{d}}_i$ and $\bar{\mathbf{d}}_j$ are encoded within different tangent frames, the later needs to be parallel-transported before being compared using $\bar{\mathbf{d}}_i - \bar{\mathrm{R}}_{i \leftarrow j}\, \bar{\mathbf{d}}_j$, where $\bar{\mathrm{R}}_{i \leftarrow j}$ is a 2D rotation encoding the in-plane rotation from j to i.

We pre-compute it as $\bar{\mathrm{R}}_{i \leftarrow j} = \mathrm{B}_i^\top \mathrm{R}_{i \leftarrow j}\, \mathrm{B}_j$ where $\mathrm{R}_{i \leftarrow j}$ is the 3D rotation aligning the normal $\mathbf{n}_j$ with $\mathbf{n}_i$. For efficiency we encode both the tangent vectors and rotations as complex numbers. In practice, the parallel-transport rotations are thus incorporated within the weighted Laplacian matrix by multiplying each off-diagonal element L_{ij} by the unit complex version of $\bar{\mathrm{R}}_{i \leftarrow j}$. We refer to [KCPS13] (their Section 2) for more details on parallel transport and an alternative computation of the transport rotations.

Finally, the linear value constraints $\bar{\mathbf{d}}(\mathbf{p}_{ij}) = \bar{\mathbf{d}}_{ij}$ on the contact-boundary $\partial\mathcal{C}$ are introduced in a least-squares sense, as described in Appendix B. A notable difference is that parallel-transport rotations from the edge extremities i and j to the point $\mathbf{p}_{ij}$ also need to be included in the constraint matrix. The result of this process is depicted in Figure 5.3.

Despite the application of the sine function before the diffusion, in some cases of deep collisions, the diffused directions still align themselves too quickly with the surface normals. Unlike our 1D parametrization, the harmonic diffusion of the directions does not produce an uniform variation of the resulting direction field: it causes fast variations of the directions close to the contact region. To obtain more uniform variations over the deformable region, we use the previously computed normalized pseudo-geodesic distance u and an additional scalar diffusion to correct the result of the direction diffusion. This additional diffusion is performed with the same type of constraints as the directions with 0 at the exact-boundary of the contact zone and 1 at the external boundary of the deformable region. We then remap the diffused projected direction according to the uniform parametrization before recovering the 3D direction. The final diffused projection for a vertex i is thus:

$$\bar{\mathbf{d}}_i = \frac{u_i}{u'_i}\, \bar{\mathbf{d}}'_i,$$

where u_i is the uniform parametrization of this vertex, u'_i is the parametrization obtained by harmonic diffusion over the deformable region, and $\bar{\mathbf{d}}'_i$ is the diffused projected

direction as computed in Equation 5.3. At the end of this process, we obtain a uniformly varying direction field from the mapping direction on $\partial\mathcal{C}$ to the undeformed mesh normal on $\partial\mathcal{D}$.

Discussion. The central question is at what speed this direction field should align itself with the normal field? According to our observations, the direction field should vary quickly toward the normals of the initial surface but not too quickly. In practice, an uniform variation of the direction field is not always desirable; this depends on several factors such as the shape and size of the deformable region $\mathcal{D}$. In some cases it would be preferable to increase the speed of convergence towards the normal field, while in other cases, it should be decreased compared to the uniform variation. We can use such a transfer function to control the speed of convergence and offer to the user some control over this parameter. However, adding another user parameter would complicate the artist's work, we thus propose a second simpler and faster method, which, in practice, does not seem to suffer from this problem.

2.2. Blended direction field

Two key ingredients emerged from the method described in the previous section: the need for considering the orientation of the input surface and the notion of variation speed towards the surface normals. Taking into account these two elements, we devised a simpler and computationally efficient two step process that explicitly controls the blending with the normal field.

We first compute a mapping vector field $\bar{\mathbf{d}}$ over the working region through a standard harmonic diffusion of the mapping directions $\hat{\mathbf{d}}_l$ defined for each contact zone $\mathcal{C}_l$ resulting from the collision of the current working region with the other working regions $\mathcal{W}_l$ (Section 2.2). More precisely, we set linear constraints of the form $\bar{\mathbf{d}} = \hat{\mathbf{d}}_l$ along each contact zone boundary $\partial\mathcal{C}_l$, and natural Neumann conditions over the remaining exterior boundary. In practice, this problem can be solved reusing the matrix factorized for solving the second Poisson problem of the previous modified heat-method, which makes this diffusion inexpensive. In our current implementation, used for generating all the results in this manuscript, this intermediate vector field $\bar{\mathbf{d}}$ is normalized. Note that this step can be by-passed in the usual case of a single object colliding with the current working region, since the result is a constant field $\bar{\mathbf{d}} = \hat{\mathbf{d}}$.

In a second step, this vector field is blended with the surface normal field using the 1D parametrization u as weights to accomplish the desired transition:

$$\mathbf{d}_i = \text{normalize}\left(w(u_i)\,\bar{\mathbf{d}}_i + (1 - w(u_i))\,\mathbf{n}_i\right), \tag{5.4}$$

with $w(u) = (1 - u^2)^6$. This weighting function enables an asymmetric and rapid transition toward the normal field.

Discussion. With the first method presented in Section 2.1, $\mathbf{d}_i$ is only C^0 continuous at the boundaries of $\mathcal{D}$ due to the harmonic diffusion. In this second method, it only

happens when multiple contact zones are involved, since $\bar{\mathbf{d}}$ is also obtained by harmonic diffusion. In practice, this did not lead to noticeable artifacts in our experiments, but a more expensive higher-order interpolation scheme could be used if needed.

Compared to the first method, this second approach is both simpler and faster. Indeed, the first method requires the factorization of a complex matrix, four times more expensive than the factorization of a real matrix. In the second approach, we can reuse the factorization of the Laplacian matrix needed for the amplitude and slope diffusion, which makes it almost "free". When a single contact zone is detected, no diffusion and thus factorization is even necessary.

In the case of surfaces with high-frequency details, it might be worthwhile to consider a smoothed normal field to guide the displacement directions. We could for example apply Laplacian smoothing to the input surface normals to obtain a set of normals $\tilde{\mathbf{n}}_i$ that would be used as a replacement for $\mathbf{n}_i$ in Equation 5.4. However, this solution would not be perfect as only a linear deformation would be applied to the details, whereas we expect that they would be rotated to follow the deformation of the surface, resulting in an even more plausible deformation. To alleviate this issue, the approach described in [BS08] could be used as inspiration. It consists in using a multi-resolution decomposition of the input surface. First the smoothed surface would be deformed using our approach and then the details would be reintroduced. However, applying this idea to our context is rather challenging because we would still have to consider the full detailed surface for handling the collision.

Finally, as previously mentioned, the intermediate vector field $\bar{\mathbf{d}}$ is normalized in our current implementation. It would be interesting to experiment with an unnormalized field as it would have the benefit of naturally degenerating into the normal field when the initial mapping directions are very far apart, or when the direction field turns in the "wrong" direction as illustrated at the beginning of this chapter.

3. Amplitude and slope fields

As defined in Equation 5.1, the profile curve $\mathcal{H}$ is parametrized by its *amplitude* a, and *slope* s at $u = 0$ such that $\mathcal{H}_{a_i,s_i}(0) = a_i$ and $\mathcal{H}'_{a_i,s_i}(0) = s_i$. Yet, we must know these two ingredients at every vertex of $\mathcal{D}$ to be able to instantiate the adequate adjusted profile curve.

At a first glance, this problem seems equivalent to the previous diffusions. There is, however, a fundamental difference that prevents the use of the same numerical scheme. In Section 1 and 2 of this chapter, the constraints on $\partial\mathcal{C}$ for the 1D parametrization u and $\mathbf{d}$ are either constant (using the second version in Section 2.2) or varying extremely smoothly, with maximal variations of the resulting fields in the direction orthogonal to $\partial\mathcal{C}$. In contrast, the value of a and s may vary greatly along $\partial\mathcal{C}$ and the resulting field is expected to be nearly constant in the direction orthogonal to $\partial\mathcal{C}$. In this context, with the use of least-squares linear constraints the maximum principle of harmonic interpolation cannot be guaranteed. Therefore, it tends to yield spurious values around vertices linked to multiple constrained edges with very different target values. We alleviate this difficulty

by explicitly computing the expected amplitude and slope of each vertex $\mathbf{p}_j$ adjacent to a contact-boundary edge ij. Those values are then interpolated over the rest of the deformable region by harmonic diffusion using natural Neumann conditions at the exterior boundary of $\mathcal{D}$. These conditions allow a good preservation of the boundary values along the gradient of u.

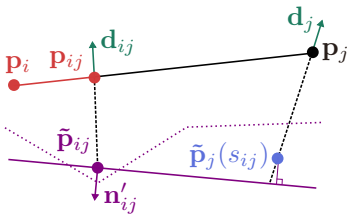
In the following, we detail how to compute the expected amplitude and slope along $\partial\mathcal{C}$.

Amplitude. For each contact boundary edge ij, the expected amplitude is given as $a_{ij} = \|\mathbf{p}_{ij} - \mathbf{p}'_{ij}\|$, i.e., the magnitude of displacement necessary to be projected on $\mathcal{S}$. The amplitude of the vertex $\mathbf{p}_j$ can then be computed as the weighted average of the displacement magnitude established in parametric space:

$$a_j = \frac{\sum_{i \in N_j} \alpha_{ij}\ a_{ij}}{\sum_{i \in N_j} \alpha_{ij}}\,,$$

where N_j denotes the set of adjacent vertices of $\mathbf{p}_j$ within the contact-zone. The weights α_{ij} ensure temporal continuity over the course of the animation.

Slope. It is worth recalling that s is the slope of the magnitude of the displacement related to $\mathcal{H}$ and not the slope of the deformed surface at the boundary of the contact zone. Therefore, s and a are linked together and cannot be computed independently. The slope field s is computed using the same weighted average for the vertices adjacent to the contact boundary. We thus need to estimate the expected slope s_{ij} at each contact-boundary edge, taking into account the values of the parametrization u, direction field $\mathbf{d}$, and amplitudes a that have been actually computed. For a small u, the profile curve $\mathcal{H}$ can be approximated as a linear function depending only on the previously computed amplitude, and the yet unknown slope. Along the edge ij, the displaced position of $\mathbf{p}_j$ can thus be approximated as a function of its slope value s_{ij}:



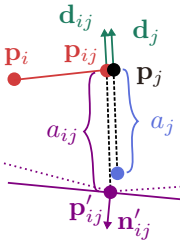
$$\tilde{\mathbf{p}}_j(s_{ij}) = \mathbf{p}_j + (-a_j + u_j\, s_{ij})\, \mathbf{d}_j\,.$$

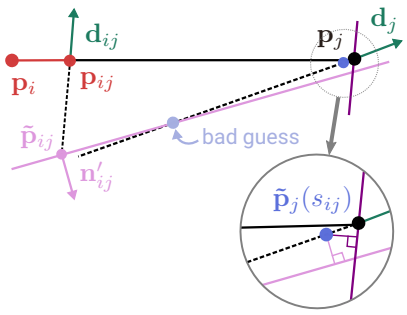
As illustrated in the inset figure, we estimate s_{ij} such that $\tilde{\mathbf{p}}_j$ lies on the tangent plane of the rigid surface:

$$(\tilde{\mathbf{p}}_j(s_{ij}) - \tilde{\mathbf{p}}_{ij})^\top \mathbf{n}'_{ij} = 0\,,$$

where $\tilde{\mathbf{p}}_{ij} = \mathbf{p}_{ij} + a_j \mathbf{d}_{ij}$, and $\mathbf{n}'_{ij}$ is the normal of the rigid surface at $\mathbf{p}'_{ij}$ obtained through barycentric interpolation.

Notice that the target plane is not positioned exactly on the rigid surface. This approximation actually improves stability in cases where u_j is extremely small (say $< 10^{-4}$) and the actual value of a_j is slightly different than a_{ij}. Such a case would lead to arbitrarily large slope estimates to cope for the tiny gap between a_{ij} and a_j.





However, as depicted in the inset figure, this problem becomes hill posed when the direction $\mathbf{d}_j$ is aligned with the edge ij, $\mathbf{e}_{ij}$. We mitigate this shortcoming by constraining $\tilde{\mathbf{p}}_j$ to lie on the plane which is the most orthogonal to the edge and passing through $\mathbf{p}_j$ and $\mathbf{d}_{ij}$ (purple line). In summary, we solve these two distance-to-plane equations in a least-square sense (blue point):

$$\begin{cases} (\mathbf{p}_j - \tilde{\mathbf{p}}_j)^\top \bar{\mathbf{n}}_{ij} = 0 \\ (\tilde{\mathbf{p}}_j(s_{ij}) - \tilde{\mathbf{p}}_{ij})^\top \mathbf{n}'_{ij} = 0 \end{cases} \quad \text{with} \quad \bar{\mathbf{n}}_{ij} = \frac{\mathbf{d}_{ij} \times (\mathbf{e}_{ij} \times \mathbf{d}_{ij})}{\|\mathbf{d}_{ij} \times (\mathbf{e}_{ij} \times \mathbf{d}_{ij})\|}.$$

4. Final deformation

At this point, almost all the ingredients required to evaluate Equation 5.1 have been computed for every vertex of the deformable region $\mathcal{D}$. In this last step, we need to combine them to instantiate the profile curve $\mathcal{H}$, defined in Section 4.1, and to compute the final positions of the vertices of $\mathcal{D}$. The volume preservation constraint presented in Section 4.2 allows us to achieve plausible deformations with artistic control.

4.1. Profile curve definition

To instantiate the deformation profiles, we use parametric open uniform cubic 2D B-splines $\mathbf{f} : t \in [0,1] \to (f_1(t), f_2(t))$ for their smoothness, local control, high flexibility, ease of editing, and ability to generate profiles with a vertical slope at $u = 0$. The downside of using a parametric curve is that the evaluation of the profile $\mathcal{H}(u)$ at a given u involves the inversion of f_1:

$$\mathcal{H}(u) = f_2(f_1^{-1}(u)).$$

In practice, the secant method turned out to be effective enough to make the cost of these inversions negligible compared to the rest of the pipeline. Alternatively we could also have used a 1D B-spline curve, which can also achieve extremely steep slopes by adjusting the knot vector, but we would have lost the local control and ease of editing properties.

As illustrated in Figure 5.4, the shape of the B-spline $\mathbf{f}$ is defined by six control points $\mathbf{v}_k$, $k \in \{0 \dots 5\}$, that are *automatically* adjusted for each vertex i of the deformed surfaces to match the given amplitude a_i, slope s_i, as well as the volume preservation and user-controlled bulging parameters h_v and γ. The definition of $\mathbf{f}$ can thus be written as:

$$\mathbf{f}(t) = \begin{pmatrix} f_1(t) \\ f_2(t) \end{pmatrix} = \sum_{k=0}^{5} \mathbf{v}_k N_k^3(t) \quad \text{with} \quad t = f_1^{-1}(u).$$

The following table summarizes the default values for the control points:

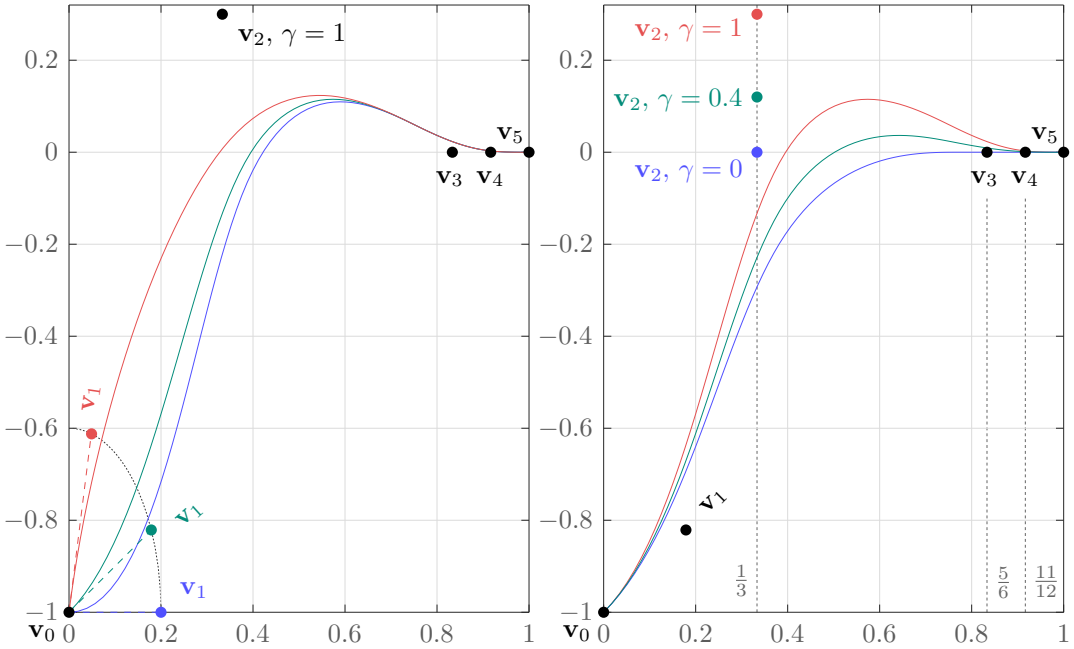
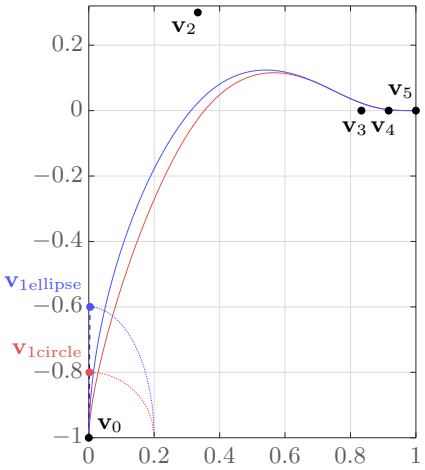


Figure 5.4.: Profile curves. The control points $\mathbf{v}_0$ and $\mathbf{v}_5$ are fixed to ensure surface continuity. The position of $\mathbf{v}_1$ is constrained to the ellipsoid arc with center $(0, -1)$, semimajor axis 0.4, semiminor axis 0.2 and fully determined by the slope s_i at the evaluation vertex i (left). The ordinate of $\mathbf{v}_2$ is optimized according to the user-defined bulging parameter γ (right). The abscissas of $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$ can be edited for additional artistic control of the bulge position and shape.

	$\mathbf{v}_0$	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$	$\mathbf{v}_5$
abscissa (x)	0	$\dfrac{0.2}{\sqrt{1 + {s_i}^2/4}}$	$\dfrac{1}{3}$	$\dfrac{5}{6}$	$\dfrac{11}{12}$	1
ordinate (y)	$-a_i$	$a_i(s_i\, x_{\mathbf{v}_1} - 1)$	$\gamma \cdot h_v$	0	0	0

The rationale behind the formulas controlling $\mathbf{v}_1$ is depicted in Figure 5.4 (left). Recall that $\mathbf{v}_1$ must be positioned to satisfy the given slope s_i, and since it can be arbitrarily large, we need to adjust both its abscissa and ordinate. To do so, we consider a canonical configuration with unit amplitude (dividing the ordinates by a_i) and we constrain $\mathbf{v}_1$ on an arc centered in $(0, -1)$. The simplest solution would have been to consider a tangent with constant magnitude, i.e., to constraint $\mathbf{v}_1$ on a circular arc. However, we found that, when the slope is expected to be very steep (nearly vertical), the difference between this linear approximation and the actual displacement mag-



nitude can be very large even if u_j is very small. We alleviate this issue by increasing the length of the tangent vector of the B-spline parametric curve according to the magnitude of the slope by adjusting the position of the second control point along an ellipse instead of a simple circle (see inset figure). This way, the profile curve better respects the linear approximation made to estimate the slopes.

The last three control points are aligned with the horizontal axis to ensure C^2 continuity at $u = 1$. To design the desired shape of the bulge, the absissa of the control points $\mathbf{v}_2$ to $\mathbf{v}_4$ can be adjusted by the user either globally or locally using brush tools. More details and examples are presented in Section 5 of this chapter and also in Chapter 7.

Discussion. To define the cubic B-spline $\mathbf{f}$, we use knots with multiplicity 4 at the boundaries of the interval $[0, 1]$ to make sure that the curve passes through $\mathbf{v}_0$ and $\mathbf{v}_5$. We thus use the following knot vector:

$$\tilde{\mathbf{t}} = (0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1)^\top$$

By definition of a cubic B-spline, the interval of influence of a control point $\mathbf{v}_j$ is $[\tilde{t}_j, \tilde{t}_{j+4}[$. Taking this information into account, we could optimize the diffusion of the amplitude and slope corresponding to the $\mathbf{v}_0$ and $\mathbf{v}_1$, respectively. Indeed, the impact of those two control points are limited to the region where $t < \tilde{t}_5$, i.e., $t < \frac{2}{3}$. However, because the boundary conditions would be put on a different boundary, we have to be aware that the results of the diffusion obtained using this optimization would be slightly different than the current results.

4.2. Volume constraint

As illustrated in Figure 5.4 (right), the ordinate of $\mathbf{v}_2$ controls the amount of bulging of the deformation. It is determined as the product of a user-controlled parameter γ allowing to either suppress or exaggerate the bulge, and a scalar value h_v which is automatically recomputed at each frame to approximately preserve the volume of the elastic object when $\gamma = 1$.

Our implementation considers an exact definition of the volume preservation constraint using the B-spline equation. We first define the displacement of each vertex as:

$$\mathbf{p}'_i - \mathbf{p}_i = \mathbf{d}_i \left(h_v N_2^3(t_i) + \sum_{\substack{k=0 \\ k\neq 2}}^{5} y_k N_k^3(t_i) \right), \tag{5.5}$$

where $t_i = f_1^{-1}(u_i)$, y_k is the ordinate of $\mathbf{v}_k$ and N_k^3 corresponds to the k^{th} cubic B-spline basis function. Here we suppose that $y_2 = h_v\gamma$, with $\gamma = 1$ since we want full volume preservation.

Then, we use this equation to express the contribution of each face of the deformed regions of the surface in the evolution of the total displaced volume. It corresponds to the signed volume of the polygon composed by the initial face and the displaced one.

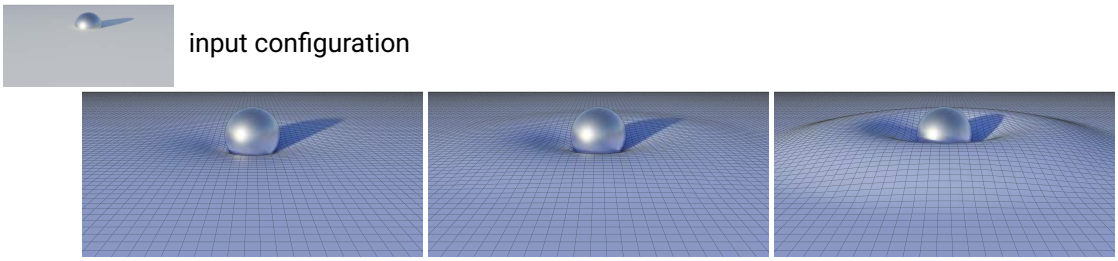


Figure 5.5.: Variations of the pseudo-volume conservation parameter γ of an elastic plane colliding with a rigid sphere. From left to right : This parameter allows the user to either cancel ($\gamma = 0$), approximately compensate ($\gamma = 1$) or exaggerate the reinjection of volume lost in the collision. [$\bar{k} = 1$, $\tau = 0, \bar{\phi} = 150$]

By summing over all the faces, this yields a cubic equation, whose smallest positive root is the expected solution, h_v. All the details are available in Appendix C.

So far, to compute the value of h_v ensuring full volume preservation, we made the assumption that $\gamma = 1$. When applying the final displacement to each vertex of the deformed surface using Equation 5.5, by varying the value of γ, the user can preserve, reduce or even exaggerate the amount of bulging as illustrated in Figure 5.5.

5. Bulge repartition

So far, the bulge is isotropically spread around the contact zone, which might not always be desirable. To spatially control the amount of bulge, a simple approach consists in letting the user paint a so called *bulge map* η introduced in the profile curve $\mathcal{H}$ as a multiplicative factor of the ordinate h_v. By taking into account this new factor during the optimization of h_v (previous section), the volume can still be globally preserved. Indeed if we take Equation 5.5 and insert the bulge map influence η_i, the displacement of each vertex i is defined by:

$$\mathbf{p}'_i - \mathbf{p}_i = \mathbf{d}_i \left((\eta_i h_v) N_2^3(t_i) + \sum_{\substack{k=0 \\ k \neq 2}}^{5} y_k N_k^3(t_i) \right),$$

The final system still boils down to a cubic equation to compute h_v. Since we are still using here $\gamma = 1$ to find h_v satisfying the volume preservation constraint, the local factor η_i is a relative quantity to allocate a portion of the volume that needs to be redistributed. In other words, multiplying all η_i by the same factor does not change the final result. Once again here, the user can still control the volume preservation when applying the final displacement by varying the value of γ.

We describe in Section 5.1 the tools that we developed to ease the edition of the bulge map. In Section 5.2, we then present an automatic method to procedurally generate such a map so that plausible anisotropic deformations are produced.

5.1. Painted bulge map

Far from being homogeneous in the real world, a bulge resulting from a collision can sometimes reveal surface details that are not visible at rest or even create wrinkles from the contact following the gradient of our 1D radial parametrization u. To grant the user with as much control as possible over the final deformation, we have added to our method the possibility to distribute spatially the bulge by directly painting η on the surface. Painting tools are commonplace many 3D modelling software, where the user uses a circular brush to assign various values to vertices, faces or edges. We offer to the artist two types of brushes with art-controllable radius, a *classical* one and a second one that takes into account the specific radial parameterization of our deformation. These brushes support different operations applicable to η: set, add, subtract or smooth. For each of these operations, the magnitude of the effect smoothly vanishes at the brush boundary to avoid any unwanted discontinuity.

Local brush. The first type of brush is a simple circular brush with which the artist can set a value of η to the vertices located inside its radius. To avoid discontinuities at the brush boundary, a fall-off can be used. It sets the maximum value at the center of the brush and its contribution smoothly decreases to 0 for the vertices located at its boundary. This brush can be used to reveal details as illustrated in Figure 5.6 as if the material of the object was more elastic at some particular locations on the surface. With this kind of tool, the user is completely free to design the details as she or he wants, which will be revealed smoothly as the volume gets distributed over the deformed surface.

We can use this type of brush to edit other parameters of the profile curve to control the position of the bulge, for example the absissa of its control points $\mathbf{v}_2$ and $\mathbf{v}_3$. However the nature of this profile and its definition around the contact zone make it difficult to produce a coherent profile. This is what motivated the development of our *radial brush*.

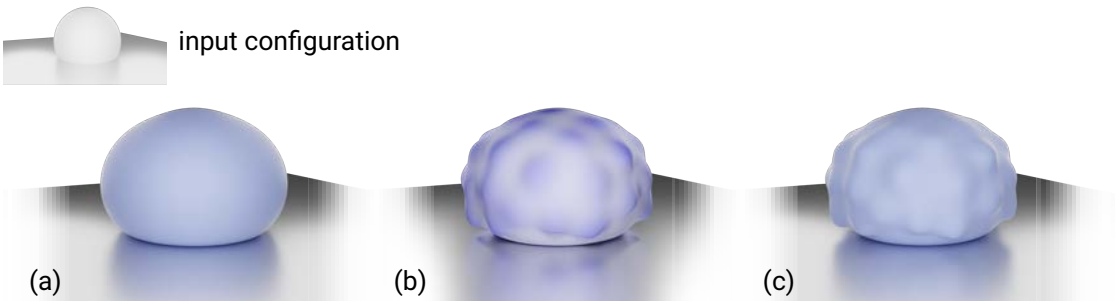


Figure 5.6.: Local Painting of an elastic sphere colliding with a rigid plane. (a) Our standard pipeline produces a uniform distribution of the bulge. (b) The bulge distribution can be guided by the bulge map η painted in levels of blue directly on the sphere, (c) resulting in a final deformation that reflects this new distribution while still preserving the overall volume.[$\bar{k}_{\text{sphere}} = 1$, $\tau = 0$, $\bar{\phi}_{\text{sphere}} = 60$]

Radial brush. The second type of brush that we developed is based on the radial definition of the deformation profile. Instead of locally modifying η at an arbitrary surface position, we set the same value to all the vertices sharing the same profile radially around the contact zone, i.e., following the gradient of u. Using the same circular brush metaphor, this modifier is applied as a two-step process. First, we search all the vertices affected by the brush, that we call the selected region $\mathcal{Z}$. Second, we compute the spatially-varying magnitude of the brush effect that must be assigned to each vertex of this selection.

The definition of the selected region is done in two main steps as illustrated in Figure 5.7 for the case of a rigid sphere colliding with an elastic plane. Starting from the position of the center of the brush projected on the surface, we follow on mesh edges the corresponding isovalue of u until we reach, in both directions, the distance equivalent to the user-specified brush radius. Doing so, we obtain a point on each boundary of the selected region $\mathcal{Z}$. From these points, we are now able to find the two lateral boundaries $\partial\mathcal{Z}_1$ and $\partial\mathcal{Z}_2$ that delimit the selection. Using the same tracing process, we follow the gradient of u until we reach the inner and outer boundary of $\mathcal{D}$, namely $\partial\mathcal{C}$ and $\partial\mathcal{D}$.During this second tracing phase, we store the position of all the boundary points on the encountered edges forming two polylines that corresponds to the lateral boundaries of the selection. The boundaries $\partial\mathcal{C}$ and $\partial\mathcal{D}$ close this region.

We could have assigned the same brush magnitude to the vertices located inside the selected region. However, to obtain a smooth deformation, we apply a fall-off to this magnitude centered between the two lateral boundaries $\partial\mathcal{Z}_1$ and $\partial\mathcal{Z}_2$. To do so, we need a 1D parametrization $\nu \in [0, 1]$ of the selected region which is orthogonal to the

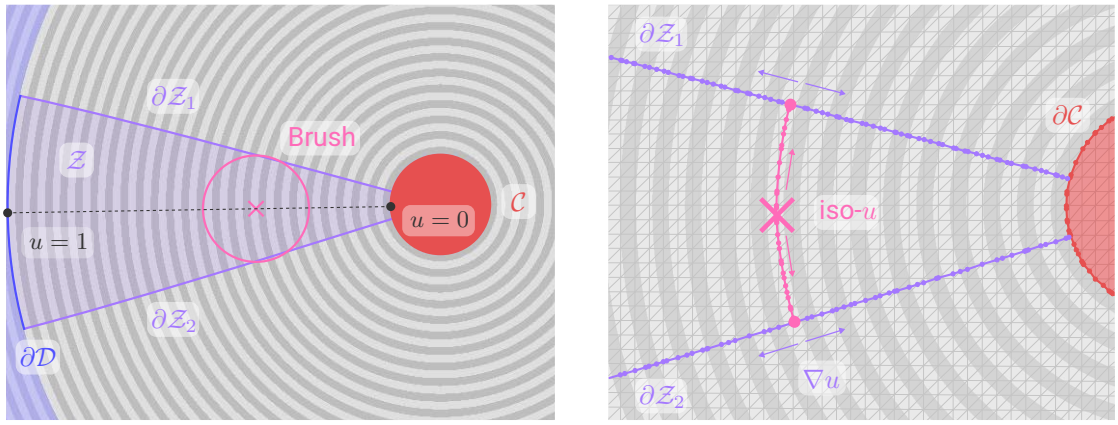


Figure 5.7.: Radial Painting of an elastic plane colliding with a rigid sphere (not shown). Left: The aim of this tool is to apply the same change on all the vertices sharing the same geodesic (dotted line for example) from the boundary of the contact zone $\partial\mathcal{C}$ to the exterior boundary of the deformable region $\partial\mathcal{D}$. The two lateral boundaries of the selected region $\mathcal{Z}$ depend on the brush center and radius set by the user. Right: To define the lateral boundary of $\mathcal{Z}$, we trace a two-sided path from the brush center following the corresponding isovalue of u and stop when reaching the radius of the brush. $\partial\mathcal{Z}_1$ and $\partial\mathcal{Z}_2$ are then defined as the polylines along ∇u until $\partial\mathcal{C}$ and $\partial\mathcal{D}$ are reached.

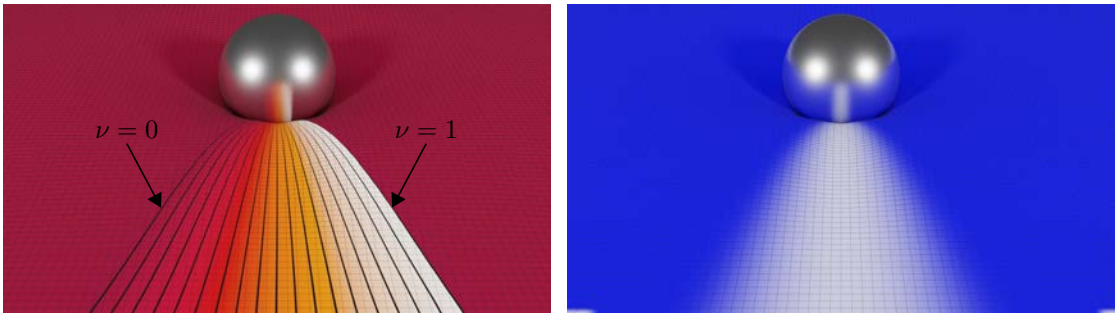


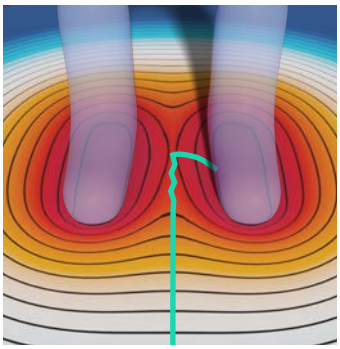
Figure 5.8.: Selected region parametrization of an elastic plane colliding with a rigid sphere. Left: the selected region $\mathcal{Z}$ is equipped with a 1D parametrization ν with Dirichlet constraint on $\partial\mathcal{Z}_1$ and $\partial\mathcal{Z}_2$ and Neumann conditions on $\partial\mathcal{C}$ and $\partial\mathcal{D}$. Left: This parametrization allows us to define a fall-off on the user-controlled brush magnitude to produce smooth deformations.

gradient of u. Using the exact boundary points defined in the first step, we compute this parametrization by harmonic diffusion with Dirichlet conditions along the boundary $\partial\mathcal{Z}_1$ and $\partial\mathcal{Z}_2$ and Neumann conditions on $\partial\mathcal{C}$ and $\partial\mathcal{D}$. As in Section 1, $\partial\mathcal{Z}_1$ and $\partial\mathcal{Z}_2$ are not defined on existing vertices of the mesh but as arbitrary polylines along its edges, we thus implement the Dirichlet boundary condition along them as a set of least-square linear constraints equal to 0 on one boundary and 1 on the other, as illustrated in Figure 5.8. The isovalues of the resulting parametrization ν are mostly aligned with ∇u and $\nu = 0.5$ roughly corresponds to the position of the center of the brush. Once the selected region is equipped with this 1D parameterization, we can modulate the user-controlled brush magnitude by the following C^1 fall-off function:

$$w(\nu) = (1 - (2\nu - 1)^2)^2.$$

In Figure 5.9 we compare the resulting deformation of the radial painting illustrated in Figure 5.8 with the standard isotropic bulge. As previously mentioned, this type of brush can also be used to edit the abscissa of $\mathbf{v}_2$ or $\mathbf{v}_3$ and still produce plausible deformations along the profile, as illustrated in Section 3 of Chapter 7.

Discussion. However, this method is limited to configurations producing a single contact zone or several ones but resulting in disconnected deformed regions. Indeed, if we take the case of multiple contact zones, the gradient of u does not always produce the expected tracing, as illustrated in the inset figure. More generally, in such a configuration it is not even clear what the selected region should look like. In general, this method is strongly related to our 1D parametrization. If the user wants more artistic freedom to edit the repartition of the bulge, the local brush is better suited.



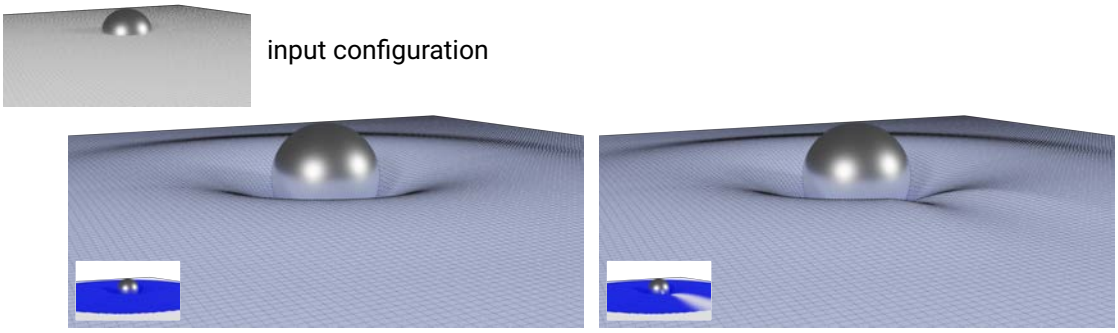


Figure 5.9.: Radial painting result on an elastic plane colliding with a rigid sphere presented in Figures 5.7 and 5.8. The corresponding bulge maps are presented on the bottom left of each result. Left: uniform distribution of the bulge. Right: the plane is deformed according to the new distribution of the bulge defined with our radial painting tools. A closer view of the bulge map is presented in Figure 5.8. Here the tool was used to reduce the bulge of the selected region, the fall-off function ensuring a smooth deformation.

5.2. Anisotropy

In addition to manual edition, the bulge map can be procedurally generated to automatically produce plausible effects. In particular, as illustrated in Figure 5.10, it is often desirable to locate the bulge in *front* of the collision (green arrow) rather than on its *sides* (red arrows). As depicted in Figure 5.10 (top-middle), we mimic this intuitive behavior by attaching to every vertex $\mathbf{p}_j$ of the contact zone a volumetric scalar field:

$$\mathcal{P}_{\mathbf{p}_j}(\mathbf{x}) = \frac{1}{\|\mathbf{x}-\mathbf{p}_j\|} \max\left(0, \frac{\hat{\mathbf{d}}\cdot(\mathbf{x}-\mathbf{p}_j)}{\|\mathbf{x}-\mathbf{p}_j\|} + \psi\right)^{\omega}$$

aligned with the mapping direction $\hat{\mathbf{d}}$ and decreasing according to both its distance to $\mathbf{p}_j$ and the cosine of the angle made between $\mathbf{x}-\mathbf{p}_j$ and $\hat{\mathbf{d}}$. This field vanishes for the points $\mathbf{x}$ lying on the plane with normal $\hat{\mathbf{d}}$ passing through $\mathbf{p}_j$. The exponent ω allows the user to balance the influence of the positional and directional terms ($\omega = 2$ in all our tests, unless specified otherwise). The parameters ψ (set to zero by default) enables the user to further reinforce the influence of the positional term: when positive, it permits to bulge even when the dot product between $\hat{\mathbf{d}}$ and $\mathbf{x}-\mathbf{p}_j$ is negative.

The bulge weight η_i of a point $\mathbf{p}_i$ of the deformation area is then obtained by summing up the contribution of each vertex in the contact zone $\mathcal{C}$ weighted by its penetration depth and associated area A_j:

$$\eta_i = \sum_{\mathbf{p}_j \in \mathcal{C}} A_j \|\mathbf{p}'_j - \mathbf{p}_j\| \mathcal{P}_{\mathbf{p}_j}(\mathbf{p}_i)\,, \tag{5.6}$$

with A_j one third of the sum of the triangle areas adjacent to $\mathbf{p}_j$. When the collision produces multiple contact zones, this sum is computed over all of them, using the associated mapping direction to evaluate $\mathcal{P}_{\mathbf{p}_j}$.

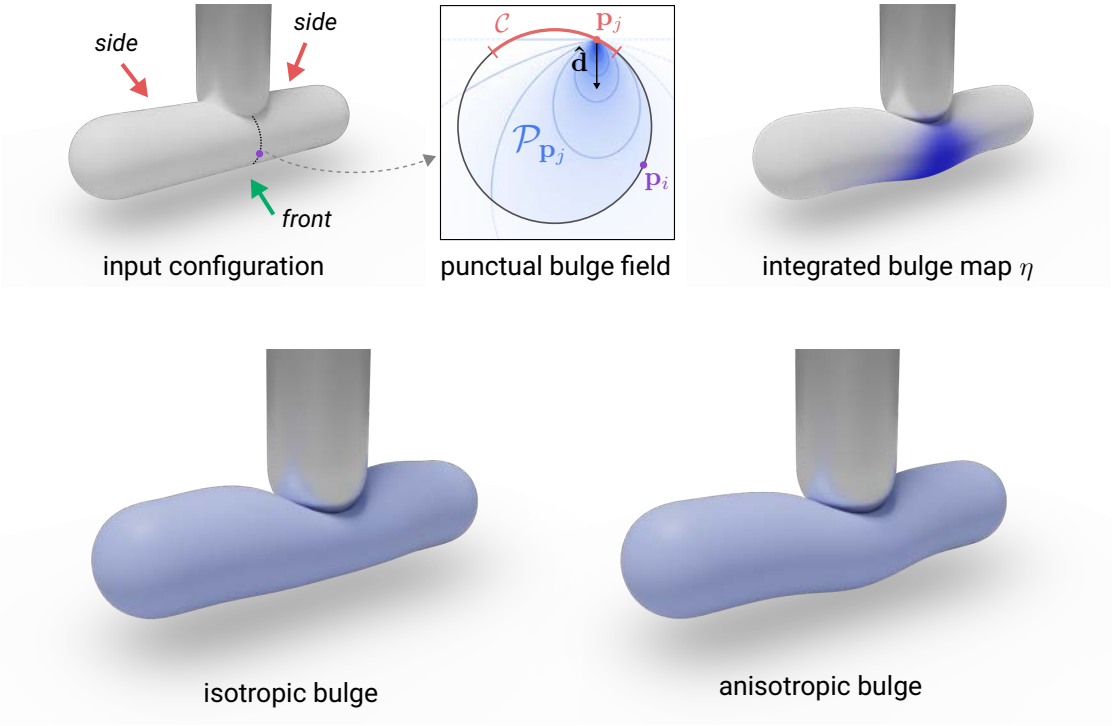


Figure 5.10.: Anisotropic bulge. Each vertex $\mathbf{p}_j$ of the contact zone $\mathcal{C}$ emits a punctual volumetric bulge field $\mathcal{P}_{\mathbf{p}_j}$ which is integrated for every point $\mathbf{p}_i$ of the deformable region to produce the bulge map η. This map is then used to anisotropically redistribute the volume of the deformation. $[\bar{k} = 1, \tau = 0]$

6. Algorithm

In this section, we summarize this chapter with an algorithm presenting the different alternatives of our pipeline. As previously mentioned, each working region $\mathcal{W}_l$ is processed independently from the others, but it can be in collision with one or multiple meshes, whose indices are stored in the set $\mathcal{J}_l$. Similarly to the previous chapter, the main algorithm on the left corresponds to the last steps of our pipeline described in [BBG21], illustrated on the right side of Figure 3.1 (steps (e) and (f)), and the blocks on the right are the alternative methods used in [BBGB20], corresponding to the left side of the same figure.

Algorithm Part 2 - Deformation

for all $\mathcal{W}_l$ **do**

Deformable region parametrization ▷ § 1

Solve $(\mathrm{id} - t\Delta)v = 0$ in a least-squares sense with the linear constraints $(1-\alpha_{ij})\,v_i + \alpha_{ij}\,v_j = 1\,,\ \forall$ edge ij crossing $\partial\mathcal{C}_l$

Compute normalized gradients $X = -\nabla v / \|\nabla v\|$

Solve Poisson equation $\Delta\phi = \nabla \cdot X$ ensuring $\phi = 0$ on $\partial\mathcal{C}_l$

$u \leftarrow \phi/\phi_{\max,l}$ ▷ $\phi_{\max,l}$ is the deformation extent

$\mathcal{D}_l \leftarrow \{\text{vertex } i \in \mathcal{W}_l \mid u_i \in]0,1]\}$ ▷ deformable region

Direction field computation ▷ § 2.2

if $|R_l| > 1$ **then**

Solve $\Delta\bar{\mathbf{d}} = 0$ with Neumann condition on $\partial\mathcal{W}_l$ and $\bar{\mathbf{d}} = \hat{\mathbf{d}}_{l,k}$ on $\partial\mathcal{C}_{l,k}\ ,\ \forall k \in \mathcal{J}_l$

else

$\bar{\mathbf{d}} \leftarrow \hat{\mathbf{d}}_{l,k}$ with $\mathcal{J}_l = \{k\}$

$\mathbf{d}_i \leftarrow \mathrm{normalize}\left(w(u_i)\,\bar{\mathbf{d}}_i + (1-w(u_i))\,\mathbf{n}_i\right),\ \forall i \in \mathcal{D}_l$ ▷ Eq. 5.4

Amplitude a and slope s fields computation ▷ § 3

Estimate the amplitude $a_{ij} = \|(1-\alpha_{ij})(\mathbf{p}'_i - \mathbf{p}_i) + \alpha_{ij}(\mathbf{p}'_j - \mathbf{p}_j)\|$, $\forall$ edge ij crossing $\partial\mathcal{C}_l$

Interpolate a over $\mathcal{D}_l$ by harmonic diffusion with:

▷ $a_j = \sum_i \alpha_{ij}\ a_{ij} / \sum_i \alpha_{ij}\,,\ \forall$ edge ij crossing $\partial\mathcal{C}_l$

▷ Natural Neumann condition on the rest of $\partial\mathcal{D}_l$

Estimate the slopes s_{ij}, $\forall$ edge ij crossing $\partial\mathcal{C}_l$

Interpolate s over $\mathcal{D}_l$ by harmonic diffusion with:

▷ $s_j = \sum_i \alpha_{ij}\ s_{ij} / \sum_i \alpha_{ij}\,,\ \forall$ edge ij crossing $\partial\mathcal{C}_l$

▷ Natural Neumann condition on the rest of $\partial\mathcal{D}_l$

Profile curves instantiation ▷ § 4

Pre-compute the first two control points defining the 2D parametric B-Spline $(f_1(t), f_2(t)) = \sum_k N_k^3(t)\mathbf{v}_k$:

for all vertex $i \in \mathcal{D}_l$ **do**

$\theta_i \leftarrow \tan^{-1}(s_i/a_i)$

$\mathbf{v}_0 \leftarrow (0,\ -a_i)$

$\mathbf{v}_1 \leftarrow (0.2\cos\theta_i,\ a_i(0.4\sin\theta_i - 1))$

$t_i \leftarrow f_1^{-1}(u_i)$

Compute η_i ▷ Eq. 5.6

Compute h_v to preserve the volume exactly ▷ degree 3 equation

for all vertex $i \in \mathcal{D}_l$ **do**

Set the ordinate of $\mathbf{v}_2$ as $\gamma \cdot \eta_i \cdot h_v$

$\mathbf{p}'_i \leftarrow \mathbf{p}_i + f_2(t_i)\,\mathbf{d}_i$ ▷ displacement

Direction field computation ▷ § 2.1

for all edge $ij \in \mathcal{D}$ **do**

$\bar{\mathrm{R}}_{i\leftarrow j} \leftarrow \mathrm{B}_i^\top\,\mathrm{R}_{i\leftarrow j}\,\mathrm{B}_j$

Interpolate $\bar{\mathbf{d}}$ over $\mathcal{D}$ ▷ App. B

by harmonic diffusion and parallel transport using $\bar{\mathrm{R}}_{i\leftarrow j}$ and the linear least-squares constraints:

▷ $\bar{\mathbf{d}}_i = \mathrm{B}_i^\top \mathbf{n}_i\,,\ \forall i \in \partial\mathcal{D}$

▷ $(1-\alpha_{ij})\,\bar{\mathbf{d}}_i + \alpha_{ij}\,\bar{\mathbf{d}}_j = \mathrm{B}_i^\top \frac{(\mathbf{p}_{ij} - \mathbf{p}'_{ij})}{\|\mathbf{p}_{ij} - \mathbf{p}'_{ij}\|}\,,$ $\forall$ edge ij crossing $\partial\mathcal{C}$

for all $i \in \mathcal{D}$ **do**

$\mathbf{d}_i \leftarrow \mathrm{B}_i\,\bar{\mathbf{d}}_i + \sqrt{1 - \|\bar{\mathbf{d}}_i\|^2}\ \mathbf{n}_i$

end

6

Application to skinning

The pipeline described so far assumes that the working regions involved in the collision are clearly separated, with no overlap, and that there are zero deformations at their external boundaries. Such assumptions, however, do not always hold. This is typically the case of a skinned articulation for which two adjacent parts are colliding with each other (e.g., elbow, knee, etc.). In such cases, the deformation response should not vanish between them and, as such, it is not even possible to define a clear frontier separating these two parts. Another example of such a configuration are the lips of a closed mouth.

In this chapter, we address this limitation by allowing adjacent working regions to overlap each other, while ensuring spatial continuity through a partition of unity blending of their respective deformation responses over the so called *shared region* $\mathcal{R}$ (depicted in purple in Figure 6.1(a)). Let β be the weights forming such a partition of unity, then the final position $\mathbf{p}'_i$ of each point in $\mathcal{R}$ is obtained as:

$$\mathbf{p}'_i = \beta_i \, \mathbf{p}'_{i,1} + (1 - \beta_i) \, \mathbf{p}'_{i,2} \; ,$$

where $\mathbf{p}'_{i,l}$ denotes the displaced position obtained by applying our pipeline to the working region $\mathcal{W}_l$.

This approach is very simple but requires additional special treatments. Most importantly, we must automatically compute a strict partitioning of the working regions so that the computation of the contact surfaces (Chapter 4) can be carried out. The partition cut should follow the temporally-varying crease of the articulation. For example, during the course of an animation, the crease of an elbow does not always correspond to the same triangle strip and slightly moves on the input surface. Therefore, we cannot ask an artist to set the location of this cut on the model as a pre-process while expecting that it would be relevant for any movement of the articulation. We thus developed an automatic method based on contour detection and the scalar field β to compute this cut on the fly at each frame of the animation. Those novel ingredients are described in Section 1.

In addition, we need to introduce some synchronization points during the computation of the deformation parameters of each working region such that the intermediate positions $\mathbf{p}'_{i,l}$ are already as close as possible to each other prior to averaging them. Those changes are detailed in Section 2 of this chapter.

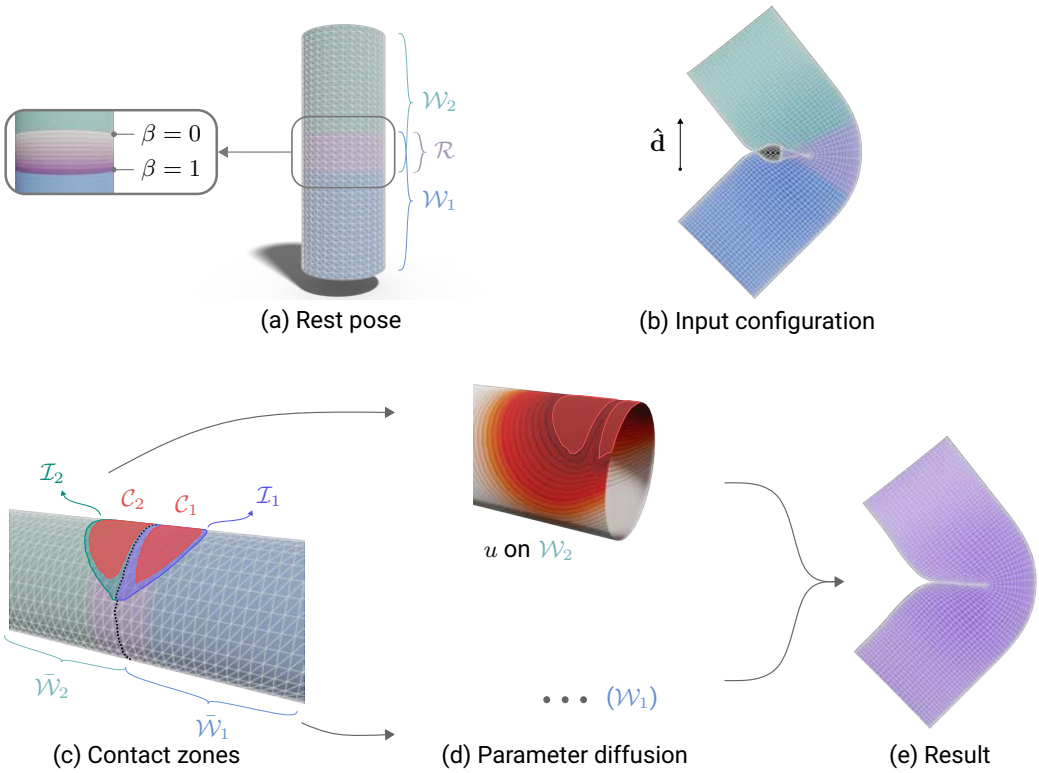


Figure 6.1.: Adjacent region overview. (a) Two overlapping regions entails a *shared region* $\mathcal{R}$ onto which partition of unity weights β are computed in a preprocess. (b) Input frame clipped to reveal the self-intersection. (c) The shared region is partitioned through an automatically computed cut (black dotted line), yielding to the non overlapping working regions $\bar{\mathcal{W}}_{l\in\{1,2\}}$ within which the contact zones are identified. (d) A deformation is then computed independently for each whole working region $\mathcal{W}_l$, starting with the diffusion of its parameters; the 1D parametrization u is shown here. (e) The final result is obtained by blending the two intermediate deformations over the shared region $\mathcal{R}$; a clipped view is used again to reveal the resolved contact and volume preserving bulge.

1. Shared region & Partition of unity

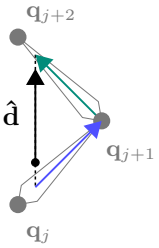
1.1. Shared region definition

The shared region $\mathcal{R}$ is defined as the set of vertices that belong to the two user-defined working regions $\mathcal{W}_1$ and $\mathcal{W}_2$. It must be large enough to always include the crease appearing when the articulation bends during the animation, and designed in a symmetric fashion such that the expected crease occurs roughly at the middle of $\mathcal{R}$.

The partition of unity weights β are precomputed on the rest pose of the skeleton by harmonic diffusion with Dirichlet boundary conditions such that $\beta = 1$ at the boundary vertices of $\mathcal{R}$ belonging only to the boundary of $\mathcal{W}_2$, and $\beta = 0$ for those belonging only to the boundary of $\mathcal{W}_1$ (Figure 6.1(a)). If $\mathcal{R}$ presents other boundaries, i.e.,vertices belonging to both working region boundaries, we use natural Neumann boundary condition for them.

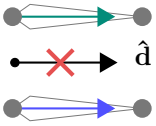
1.2. Mapping direction

The heuristic described in Section 2.2 of Chapter 4 to compute the mapping direction does not apply in the current setting, because it requires strictly separated intersection regions. In addition, as detailed in the next paragraph, this mapping direction $\hat{\mathbf{d}}$ must partition the common intersection region. To work around this chicken-and-egg problem, it is usually possible to estimate $\hat{\mathbf{d}}$ from the relative motion between the two parts of the considered articulation and, in some cases, it is even possible to instantaneously estimate it from the skeleton. For instance, in the common case of two bones connected at a joint (see inset), we compute $\hat{\mathbf{d}}$ as the vector orthogonal to the bisector of the bones lying within their supporting plane, which seems the natural direction to resolve the intersection within the fold.



More precisely, we use the transformation of the joints involved in the considered articulation. In the example illustrated by the inset figure, three joints are connected by two bones. The mapping direction is simply computed from the positions $\mathbf{q}_j$ of the three consecutive joints as:

$$\hat{\mathbf{d}} = \frac{(\mathbf{q}_{j+1} - \mathbf{q}_j) + (\mathbf{q}_{j+2} - \mathbf{q}_{j+1})}{\|(\mathbf{q}_{j+1} - \mathbf{q}_j) + (\mathbf{q}_{j+2} - \mathbf{q}_{j+1})\|}$$



Certain configurations of the skeleton do not allow to determine $\hat{\mathbf{d}}$ using this approach, see for instance the rig of a mouth in the inset figure. In that case, the two involved bones are not adjacent in the skeleton tree structure and share the exact same transformation. The previously described heuristic would thus produce an horizontal direction whereas a vertical direction is expected here. To address this limitation, we could have used the influence of the instant velocity of the respective bones. Although this idea sounds promising, we did not pursue it further, as a manual or keyframed direction was sufficient for the time being, similarly to the general pipeline.

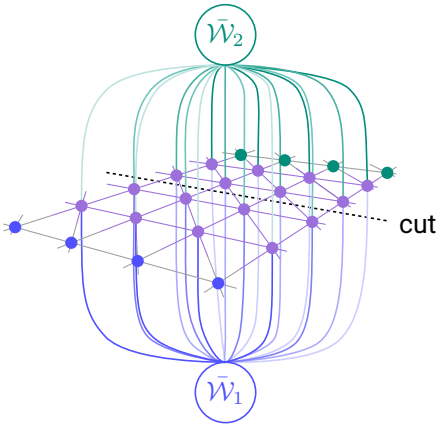
1.3. Crease detection & Partitioning

In the typical scenario considered in this section, the intersection region of each working region is expected to be open with a common portion lying in the shared region (Figure 6.1(b-c)). To be compatible with the rest of our pipeline, this shared intersection region needs to be explicitly partitioned. This partition is crucial for the mapping step of our pipeline (Chapter 4, Section 2). Indeed, we can easily imagine that the vertices of an arm have a mapping direction opposite to those of the forearm. We thus need to know which direction corresponds to which vertex to be able to perform the mapping of the surfaces. Moreover, such an explicit partionning in two distinct meshes allows us to directly reuse our simple collision detection mecanism of Chapter 4, Section 1 without having to deal with self-intersections.

More formaly, our objective is then to define a partition of $\mathcal{W}_1 \cup \mathcal{W}_2$ resulting in $\bar{\mathcal{W}}_1 \cup \bar{\mathcal{W}}_2$, where $\bar{\mathcal{W}}_1 \cap \bar{\mathcal{W}}_2 = \emptyset$. $\bar{\mathcal{W}}_l$ thus corresponds to a truncated set of $\mathcal{W}_l$. In

other words, we want to compute a cut through $\mathcal{R}$ producing two sets of vertices labeled respectively as $\bar{\mathcal{W}}_1$ and $\bar{\mathcal{W}}_2$. For instance, on the elbow example, one can intuitively imagine that the vertices of the arm should have an opposite mapping direction to those of the forearm. The partition determines where the switch between those two directions should happen to define a correct mapping.

Intuitively, the partition cut should follow the crease produced by the input deformation (e.g., the skinning). This cut is expected to slightly vary throughout the animation, and must thus be computed at runtime.



We identify this cutting crease using a graph-cut optimization [BK01] over the shared region. More precisely, we use a minimum cut/maximum flow algorithm that consists in an energy minimization to label the nodes of a graph data structure (e.g., to segment the pixels of an image). In our case, we construct this graph over the shared region $\mathcal{R}$: the nodes of the graph corresponds to the vertices of $\mathcal{R}$ and the edges of the graph to the mesh edges. For the purposes of the algorithm, we add two terminal nodes (usually called "source" and "sink") corresponding to the set of labels that can be assigned to the vertices, i.e., $\bar{\mathcal{W}}_1$ and $\bar{\mathcal{W}}_2$.The associated graph is illustrated in the inset.

The energy minimized by this graph-based method is:

$$E_{\mathcal{R}}(\mathbf{w}) = \sum_{i \in V} D_i(w_i) + \sum_{(i,j) \in N} K_{i,j}\ \delta_{w_i \neq w_j}\ , \qquad \text{with } \delta_{w_i \neq w_j} = \begin{cases} 1\ , \text{ if } w_i \neq w_j \\ 0\ , \text{ if } w_i = w_j \end{cases} , \tag{6.1}$$

where V corresponds to the vertices and N to the edges of $\mathcal{R}$, $\mathbf{w} = \{w_i |\ i \in V\}$ is the labeling vector of the vertices of $\mathcal{R}$, D_i is a penalty function and $K_{i,j}$ is the interaction potential. Typically, the regional term D_i indicates individual label-preferences of a vertex. The interaction potential $K_{i,j}$, also called boundary term, encourages spatial coherence by penalizing discontinuities between neighboring vertices. Generally $K_{i,j}$ is large when vertices i and j are similar, and $K_{i,j}$ is close to zero when they are very different. We design these two functions to obtain the desired cut of the shared region as follows.

First, since the two intersection regions should face each other along the mapping direction $\hat{\mathbf{d}}$, the cut is expected to lie along the silhouette of $\mathcal{R}$ as seen from $\hat{\mathbf{d}}$. In practice, for all vertices i and j of $\mathcal{R}$ sharing an edge e_{ij}, the boundary term is computed as:

$$K_{i,j} = 0.1 + \min_{\mathbf{x} \in e_{ij}} |\mathbf{n}(\mathbf{x}) \cdot \hat{\mathbf{d}}|\ ,$$

where $\mathbf{n}(\mathbf{x})$ denotes the surface normal at the point $\mathbf{x}$ within the edge e_{ij}. Since $\mathbf{n}$ is obtained by linear interpolation along an edge, computing this minimum is straightforward:

it is either 0 if a silhouette crosses the edge, or it is reached at one edge extremity.

Second, to regularize the previous criterion, we assume that the cut should most likely pass nearby the iso-contour $\beta = 0.5$ of the shared region. Within a graph-cut framework, this is easily implemented by using β as regional term. For every vertex i, the regional term is computed as:

$$\begin{cases} D_i(\bar{\mathcal{W}}_1) = \beta_i \\ D_i(\bar{\mathcal{W}}_2) = 1 - \beta_i \end{cases}$$

As depicted in Figure 6.1(c), the minimization of Equation 6.1 results in the segmentation of the shared region vertices into two parts, each of them forming a new truncated working region $\bar{\mathcal{W}}_l$.

2. Pipeline adaptations

Once the working regions have been partitioned for the current frame, we still need to slightly modify the pipeline described in the previous chapters to handle articulations. First, we need to compute consistent contact zones over the two working regions, as a subset of their vertices is shared. Second, to avoid discontinuity between the previously defined partitions, we must define a smooth transition of the mapping direction that are opposite on each side of the cut

2.1. Consistent contact zones and scalar fields

The computation of the contact surfaces (Chapter 4) is carried out on the truncated working regions $\bar{\mathcal{W}}_l$ without any modification. As explained earlier, the deformation should not stop at the previously defined cut, especially around the bending area of the arm for example. Since the vertices are shared in this part of the input mesh, they must also be subject to the same constraints, typically the fixed displacement and normal imposed by the contact zone.

Therefore, after their detection, we need to ensure that the identified contact zones are properly reported on the parts of the opposite working region that have been temporarily cut out. For instance, in Figure 6.1(c-d) the region $\mathcal{C}_1$ identified on $\bar{\mathcal{W}}_1$ is reported to $\mathcal{W}_2$. Once the contact zones have been determined, we no longer need to consider the truncated working regions, we use the initial ones $\mathcal{W}_1$ and $\mathcal{W}_2$, each of them including the shared vertices of $\mathcal{R}$.

All the scalar diffusions are then performed independently on each working region even on the shared vertices of $\mathcal{R}$, which explains the need for the partition of unity blending of the final displacements. Indeed, there is no guarantee to have exact correspondences between the two working regions considered in this shared subset of the initial surface.

2.2. Consistent mapping direction

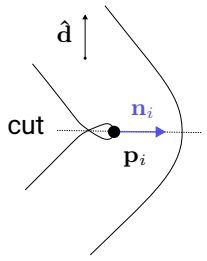
Finally, we need to update the intermediate mapping direction field $\bar{\mathbf{d}}$ such that it smoothly transitions from $\bar{\mathbf{d}}$ to the opposite direction $-\bar{\mathbf{d}}$ over the shared region. This

is accomplished using a linear interpolation parametrized by β:

$$\bar{\mathbf{d}} \leftarrow (2\beta - 1)\,\bar{\mathbf{d}}\,.$$

Note that the unnormalized vector field $\bar{\mathbf{d}}$ then vanishes at $\beta = 0.5$, and the final displacement direction $\mathbf{d}$ will thus gently fallback to the normal field according to Equation 5.4, recalled here for a vertex i:

$$\mathbf{d}_i = \text{normalize}\left(w(u_i)\,\bar{\mathbf{d}}_i + (1 - w(u_i))\,\mathbf{n}_i\right)\,,$$



with $w(u) = (1 - u^2)^6$.

If we would have normalized $\bar{\mathbf{d}}$ after the linear interpolation, it would have superseded the normal for small values of u according to the above equation. In addition to not having the expected direction of displacement, this could introduce an unwanted tangential component that may result in visual artifacts. For instance, let us consider an elbow articulation, as illustrated in the inset figure. Intuitively, on this example the displacement of all the vertices located along the silhouette zone as seen from $\hat{\mathbf{d}}$ should be defined along an horizontal direction that corresponds to the normal at this location. In particular, the exit of the fold represents a critical region for the quality of the deformation. Inside the intersection, the point $\mathbf{p}_i$ is the extreme case as it is close to the contact zones and thus corresponds to a small parameter value u_i. According to Section 2.2 with a normalized $\bar{\mathbf{d}}_i$, the final direction would not have fallen back to the normal field.

The result of this extension of our pipeline to handle skinning configurations is illustrated in Figure 6.2. We can notice the clean contact and the subtle bulge producing a final smooth deformation.

3. Discussion

Partitioning. The main limitation of the approach described in this chapter is the need for a 1D parameterization β. As it is used by the MinCut/MaxFlow algorithm, the middle of the shared region, corresponding to $\beta = 0.5$, must be roughly aligned with the expected cut. This is important not only for the definition of the cut, but also for all the blending operations applied to the vertices of the shared region. We are thus facing a chicken and egg problem when computing the cut: on the one hand the cut is defined using β and on the other hand $\beta = 0.5$ should be aligned on the cut. To avoid using this scalar field to compute the cut, one could alternatively use a random walker algorithm [Gra06] solely based on the silhouette criterion. This approach shares some similarities with the MinCut/MaxFlow algorithm. This is a graph-based segmentation method with pre-labelled nodes considered as seeds. In our case, the seeds would be the vertices that unambiguously belong to $\bar{\mathcal{W}}_l$. In practice, it boils down to an anisotropic diffusion problem, with Dirichlet conditions located on the seeds and soft diffusion barriers along the expected cut, that is, along the silhouette. If we take the same formalism as

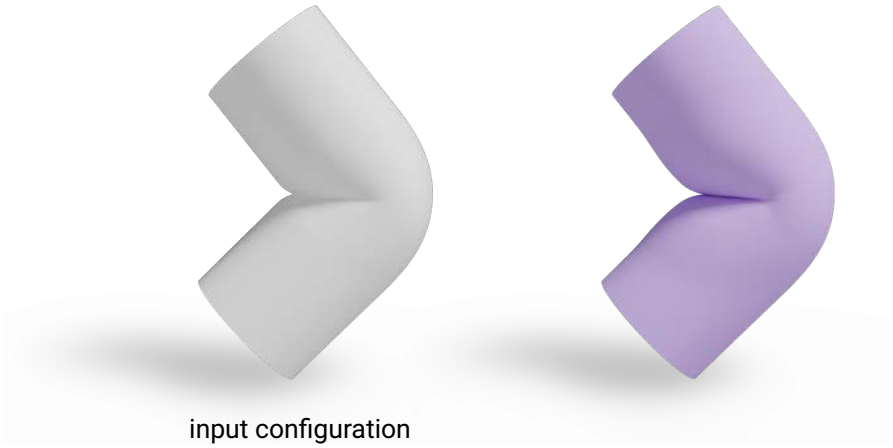


Figure 6.2.: Result of the deformation of a bending capsule with adjacent working regions corresponding to the model of Figure 6.1.

in Section 1, here we want to determine for each vertex of the input mesh the probability to belong to one or the other truncated working region $\bar{\mathcal{W}}_l$. The final segmentation is obtain by binarization with respect to 0.5 of the vector $\mathbf{w}$ containing those probabilities. More formally, if we consider Equation 6.1, the energy minimized by this random walker algorithm would become:

$$E_{\mathcal{R}}(\mathbf{w}) = \sum_{(i,j)\in N} K_{i,j}\ (w_i - w_j)^2\ ,$$

where w_i corresponds to the probability to belong to one type of seeds. With such a formulation β would be no longer necessary for the cut, but we still want the iso-contour $\beta = 0.5$ roughly aligned with the cut for the two partition of unity blends of the directions and final displacements, which is not guaranteed by neither the MinCut/MaxFlow nor the random walker algorithm.

For the sake of simplicity, when we begun to work on skinned articulations, we strived to minimize the changes and adaptations of our existing pipeline. It motivated our choice to work with two distinct working regions overlapping each other. This approach forces us to blend the displacements using β because the fields diffused on each working region are not fully consistent with each other. But what if we see those two working regions as a unique mesh instead? In this case, assuming that self-intersections are handled by the intersection detection procedure, we would still have to compute the runtime cut with the random walker algorithm to properly assign a mapping direction to each side and to extract the contact regions. Then, the idea would be to perform all the diffusions (i.e., heat-method, direction, amplitude, and slope) only once over the unique mesh instead of performing them separately on each of the two working regions. To ensure that the direction field properly fallbacks to the normal field near the silhouette seen from $\hat{\mathbf{d}}$ without using a blend factor, we would have to use an unnormalized intermediate vector field $\bar{\mathbf{d}}$ as discussed in Section 2.2 of Chapter 5. This unified mesh approach would

not only avoid the blending of displacements at the end of the process, but it would also guarantee the consistency of the resulting fields in the shared region without using β.

Input skinning deformation. Finally, although this extension of our pipeline is, in theory, agnostic to the geometric skinning method used as a preprocess, it is based on the assumption that the input mesh resulting from the skinning deformation presents a clean intersection. This goes against the objective of today's geometric skinning methods which seek a compromise between the size of the intersection and the visible deformation quality. In practice our method is thus sensitive to the quality of the input deformation.

7

Results

In this chapter we will present the results generated by our method. We will first show some comparisons with a finite element simulation, but also comparisons between the different versions of the method presented in the previous chapters. Throughout this manuscript, one of the main guidelines was artistic control. In Section 3, we will list and illustrate the various user controllable aspects of our method. Inevitably, our method comes with limitations. We will discuss in Section 4 some corner cases that illustrate those. Finally, in Section 5, we will report the computation time achieved by our method and future directions to improve them even further.

We also invite the reader to watch the videos accompanying our two publications [BBGB20, BBG21] to see these results in motion. To make the description clearer, we will call "method A" the approach associated with the left side of Figure 3.1 (i.e., [BBGB20]), which is limited, due to the ball testing algorithm, to the asymmetric case of the contacts between a rigid and an elastic object. "Method B" will correspond to the right side of the same figure (i.e., [BBG21]), which handles the more generic case of elastic-elastic contacts.

Development framework

The code developed during this thesis consists in a plugin for **Radium**, a research 3D engine for rendering, animation and geometry processing developed by the **STORM** research group of the IRIT laboratory in Toulouse, France. This plugin is implemented in C++ using various libraries for the different blocks of our pipeline. First, we used the **OpenMesh** half-edge data structure to store and manipulate triangle meshes. As previously mentioned, we use **Embree** in *robust* mode to compute segment-triangle intersections. The implementation of the Max-Flow algorithm of [BK01] used for the segmentation of the shared region in skinning configurations(Chapter 6) is available on Vladimir Kolmogorov's website. Finally, for the resolution of the numerous diffusions throughout our pipeline, we used the simplicial Cholesky solver of the **Eigen** library.

1. Qualitative results

1.1. Simple configurations

To begin with, we evaluate method A on a few test-cases made of various combinations of sphere and capsule objects. Figure 7.2 shows five basic, yet challenging, configurations with each time a pair of rigid and elastic objects in intersection followed by the result of applying our deformer. We use $\gamma = 1$ throughout. In the last two columns, we have inverted the roles of the objects, which has a notable effect on the size of the contact zone: it is small in the first instance, and large in the second instance, which illustrates its dependency on the geometric configuration. The accompanying video of [BBGB20] shows animated versions for all five configurations.

We then demonstrate the deformer of method A applied to a skinned character (Figure 7.1) in a few animation sequences of the same video. What is noteworthy here is the temporal continuity of the deformation while the character is animated. This video also shows an interactive session where the character pose is modified while the deformer is applied live, which is only made possible by the time-independency of our approach.

A last example in Figure 7.3 illustrates the deformation of an elastic arm by the rigid handle of a heavy bucket. Please note the subtle bulge of the arm around the clean contact produced by method A.

Finally, in Figure 7.4 we present some results obtained with method B on simple elastic-elastic configurations. Once again, we use $\gamma = 1$ in all these examples; the other parameters are specified directly in the figure. For the case of the cube colliding with the bumpy surface, we edited the abscissa of the control points $\mathbf{v}_2$ and $\mathbf{v}_3$ to move the bulge closer to the contact zone. This later includes the entire base of the cube. In the third example with the two aligned capsules, we can observe the symmetry of the deformation when the considered object have the same geometry and physical properties.

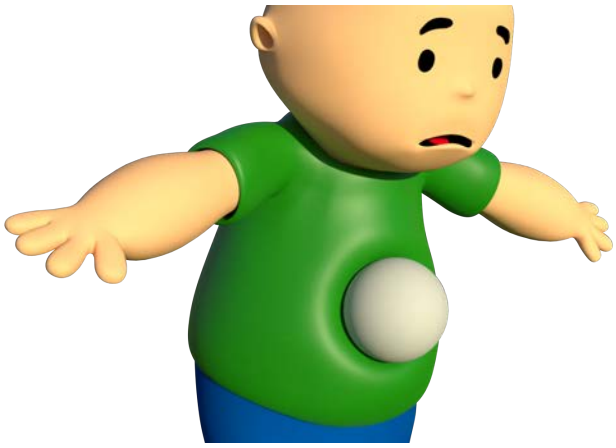


Figure 7.1.: Animated input. Even though the torso of the character is animated using linear blend skinning, our method still manages to compute a clean contact with the rigid sphere since it is applied as a post-process to geometric skinning. [Method A : $r_s = 140$, $\bar{\phi} = 300$, $\gamma = 0.5$]

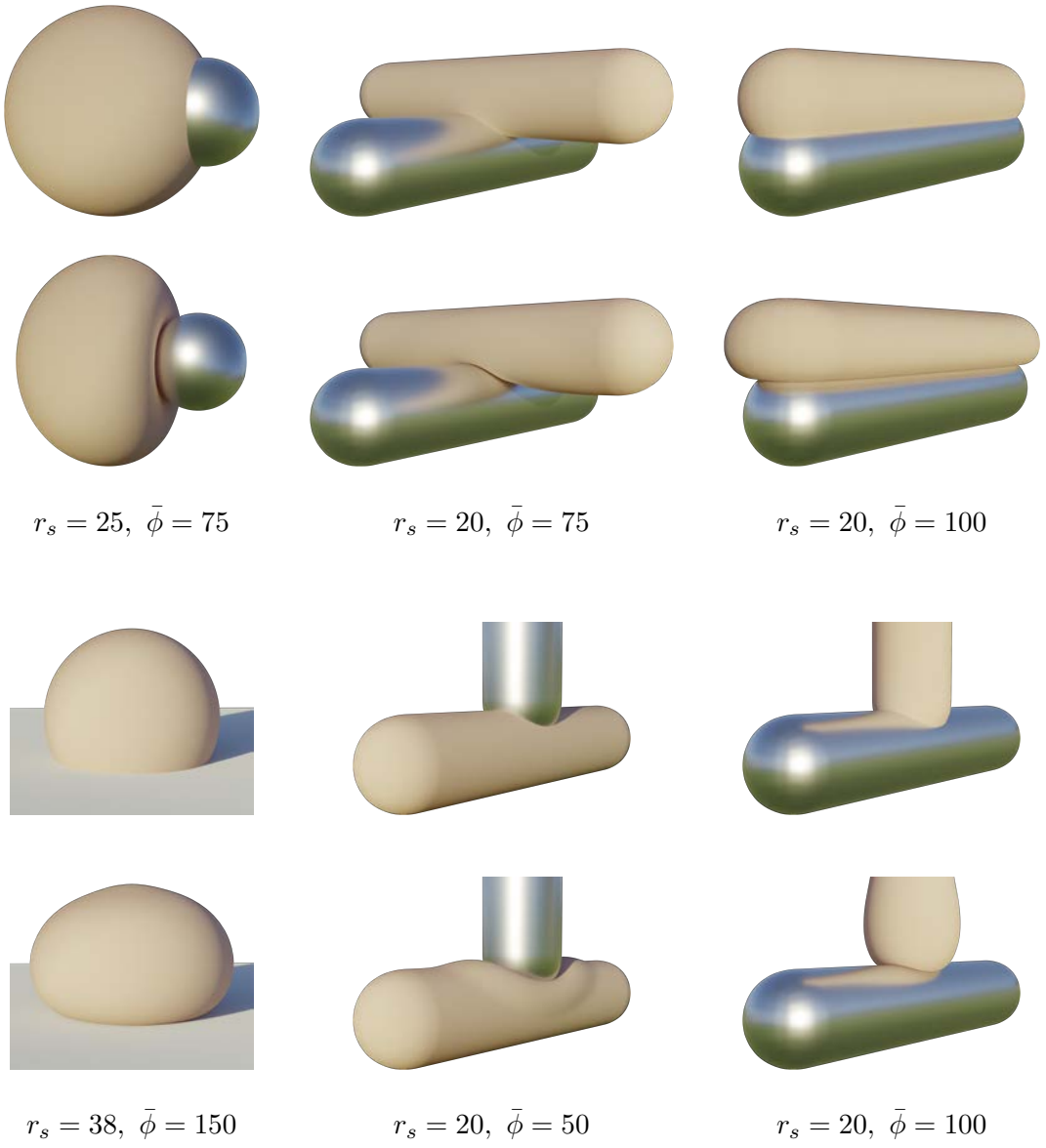


Figure 7.2.: Test examples using method A. First and third rows: objects in intersection. Second and fourth rows: result of our deformer with default settings. From left to right: A rigid sphere (diameter=50) on an elastic one (diameter=100); two horizontal capsules (diameter=100) colliding perpendicularly and parallelly; an elastic sphere (diameter=100) on a rigid plane, two capsules with different orientations, each taking the role of the elastic object in turn.

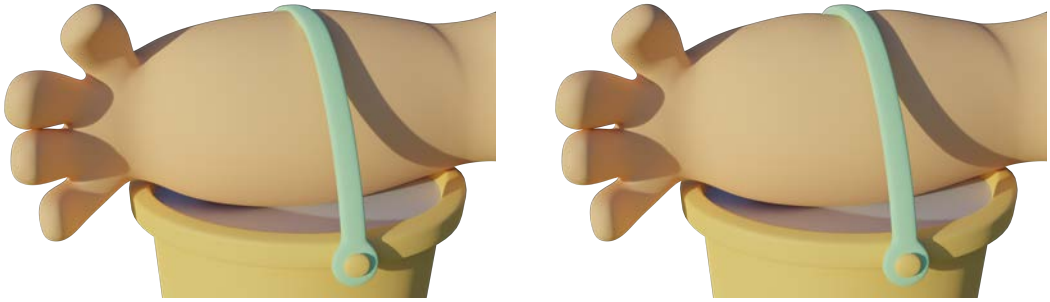
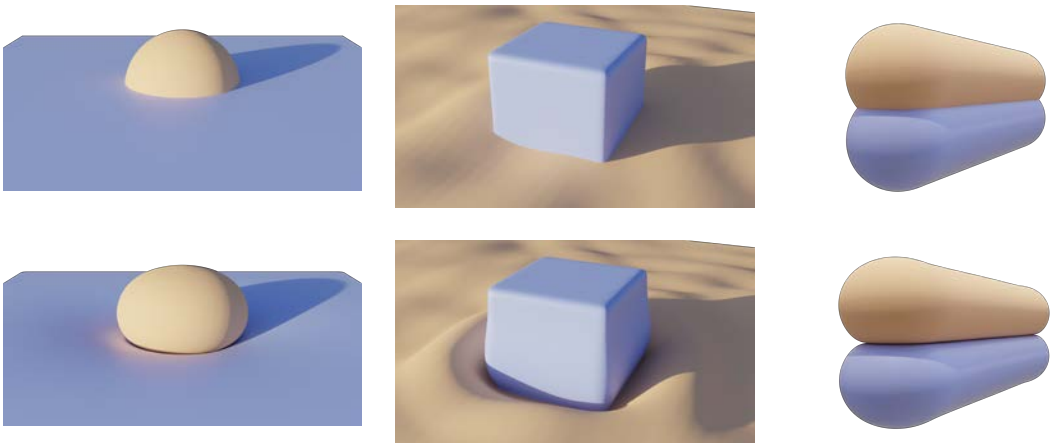


Figure 7.3.: Collision between a rigid bucket handle pressing on a chubby elastic arm. [Method A : $r_s = 30$, $\bar{\phi} = 50$].



$\varepsilon_c = 0,\ \bar{\phi}_{\text{plane}} = 150,$
$\bar{\phi}_{\text{sphere}} = 60,\ \bar{k}_{\text{sphere}} = 0.6$

$\varepsilon_c = -0.6,\ \bar{\phi} = 100,$
$\bar{k}_{\text{cube}} = 0.6$

$\varepsilon_c = 0,\ \bar{\phi} = 200,$
$\bar{k} = 0.5$

Figure 7.4.: Test examples using method B. First row: objects in intersection. Second row: result of our deformer with default settings. From left to right: A sphere colliding with a plane; a deformable cube pressing on a bumpy plane; two horizontal capsules colliding parallelly.

1.2. Multiple disconnected components

In this section, we will focus on method B when dealing with multiple soft components which requires some specific treatments presented in Section 2.2 of Chapter 5. We recall that these treatments are required to allow the definition of an independent mapping direction for each disconnected component of the contact zone.

Figures 7.5 and 7.6 show two examples of this kind of configuration: finger tips squashing an elastic ball and pressing onto the palm of its own hand. In these examples, the whole ball and the palm form a single working region. Yet the contact zones are determined independently for each finger tips, thus allowing each contact to be established with its own mapping direction and extent threshold ε_c, both being automatically computed.

Figure 7.5.: Multiple disconnected components. Left : input configuration, each finger produces an independent working region. Right: resulting deformation. Notice the subtle bulge of the finger tips and palm (e.g., around the little finger and between the index and middle fingers). [Method B : $\tau_{\text{palm}} = 0$, $\bar{\phi}_{\text{palm}} = \bar{\phi}_{\text{fingers}} = 1$, $\omega = 0$, $\bar{k}_{\text{fingers}} = 0.8$]

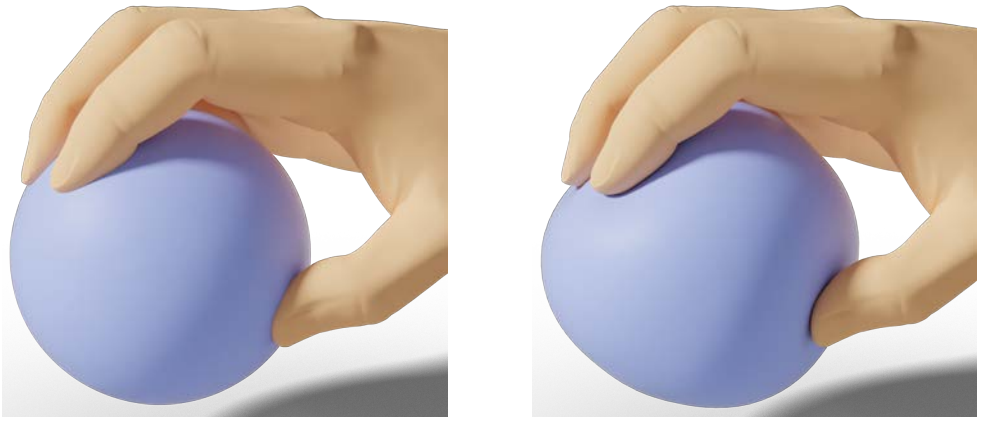


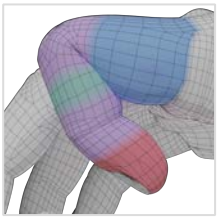
Figure 7.6.: Multiple disconnected components. Left column: input configuration, each finger produces an independent working region. Right: resulting deformation. Notice the subtle bulge of the finger tips (e.g., the thumb on the ball). [Method B : $\tau_{\text{ball}} = 0.6$, $\bar{\phi}_{\text{fingers}} = 1$, $\bar{\phi}_{\text{ball}} = 3$, $\gamma_{\text{ball}} = 3$, $\bar{k}_{\text{fingers}} = 0.8$, $\omega = 0$]

1.3. Skinning & Self-intersections.

Figure 7.8 illustrates the effects of our extended pipeline (Chapter 6) in the case of self-intersections occurring at the joint of an articulation with linear blend skinning. In the accompanying video of [BBG21], we present an extended animation showing the contact handled at the two consecutive articulations of the same finger. In this case, each phalanx corresponds to a different working region and shared regions are defined around each joint.

A slightly more complex example is shown in Figure 7.7 with self-intersections occurring between the two lips of a closed mouth animated using linear blend skinning. Whereas our automatic mapping direction heuristic works perfectly well for the finger, it does not apply to the mouth which is not an articulation. We thus manually set the mapping direction to a constant vertical vector. In this particular example, the challenge is the ceaseless variation of the position of the cut detected by our method. It is located at the corner of the mouth and not directly related to animation of the bones. This issue is exacerbated by the low sampling of the surface.

Figure 7.7.: Skeletal skinning. Our method handles elastic-elastic contacts such as in this example of a skinned mouth; the self-intersecting input configuration is shown in the background.[Method B : $\tau = -1$, $\bar{\phi} = 60$, $\bar{k} = 0.5$, $\hat{\mathbf{d}} = (0.1, -0.9, 0)$]



input regions

Figure 7.8.: Skeletal skinning. Left: input configuration and corresponding working regions. The shared regions (in purple) are between the three phalanxes. Each phalanx represents a different working region. Right: resulting deformation. [Method B : $\tau = -1$, $\bar{k} = 0.5$]

1.4. Surfaces with complex reliefs.

In Figures 7.9 and 7.10, we stress the robustness of our algorithms to the case of surfaces exhibiting relief details. First, in Figure 7.9, we present a configuration handled by method A of a rigid three-teeth comb sliding on an elastic Stanford Bunny. Here, we have three different intersection zones, each of them contributing to the same deformable region. The result illustrated in Figure 7.10 has been obtained as is by method B, with an automatically computed mapping direction (Chapter 4, Section 2.2). As the motion of the jelly is rather tangential to the surface of the Bunny, the contact response could be improved by exploiting the relative motion of the two objects or keyframing the mapping direction. Besides, for surfaces exhibiting even more intricate details, it may be wise to precompute and use a smoothed normal field to drive the displacement directions in Equation 5.4, as discussed in Chapter 5, Section 2.2.

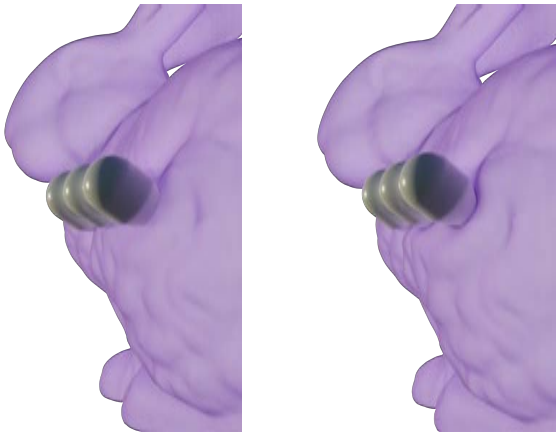


Figure 7.9.: Surfaces with complex reliefs and disconnected contacts. Left: input configuration of a comb sliding on the back of a bunny. Right: Despite the complex relief, the final deformation is clean even between the teeth of the comb. [Method A : $r_s = 32$, $\bar{\phi} = 110$]

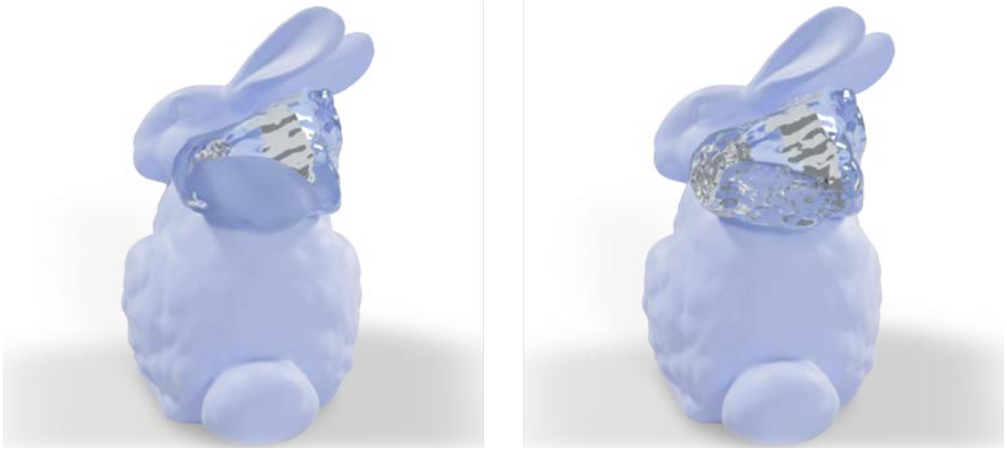


Figure 7.10.: Surfaces with complex reliefs. Left: input configuration. Right: resulting deformation. The jelly is clipped to show the resolved collision. [Method B : $\tau = -1$, $\bar{\phi}_{\text{bunny}} = 300$, $\bar{\phi}_{\text{jelly}} = 560$, $\bar{k}_{\text{jelly}} = 0.6$]

2. Comparison

2.1. Physical simulation

In order to evaluate the plausibility of our results, we tried here to reproduce the deformation obtained by a physically realistic simulation based on the finite elements method.

In Figure 7.11, we compare method A to a FEM simulation in Houdini©SideFX. As a baseline, we also show the deformation produced by the iCollide procedural deformer provided as a plugin for Autodesk Maya©. In the case of the simulation, we have made an elastic sphere fall on a rigid plane until it bounces off it. We have then matched by hand the translation motion of the sphere so that its highest point is matched with the highest point in the simulation. The parameters of both iCollide and our deformer (constant throughout the animation) have finally been adjusted manually to visually match the result of the simulation.

The parameters used for the different techniques in Figure 7.11 are summarized below:

Houdini FEM solver on a sphere of radius 1

Tet embedding	**Physics**
max tet scale: 0.06	*stiffness multiplier*: 1000
min triangle scale: 0.03125	*damping ratio*: 0.8
discretization max resolution: 1024	*mass density*: 1000
	shape/volume stiffness: 100/642
	friction: 0.1

iCollide deformer

Global parameters	**Bulge shape**
bulge: 10	*max range*: 10
radius: 50	*intersection range*: 5
offset: 0	*bulge parametric position*: $(0.197, 1)$

Method-A

Global parameters	**Profile shape**
pseudo-stiffness r_s: 25	$\mathbf{v}_2.x$: 0.2
deformation extent $\bar{\phi}$: 60	$\mathbf{v}_3.x$: 0.77
bulge $\gamma = 0.5$	

Figure 7.11.: Comparison of method A with a simulation and a procedural deformer. Center: FEM simulation in Houdini©SideFX. Left: iCollide procedural deformer in Autodesk Maya©. Right: Method A.

As seen in the video accompanying [BBGB20] where the comparison is shown in motion, our deformer is able to faithfully match the deformations and bulging effects observed in the simulation. iCollide produces obviously different results, since the contact zone corresponds exactly to the intersection region: its extent is fixed and cannot be user-controlled.

In Figure 7.12, we compare method B to the same FEM simulation method of Houdini© SideFX applied on two colliding elastic balls. In the case of the simulation in this example, we have imposed opposite forces to each elastic sphere and disabled gravity in order to provoke their collision. We proceed as described in the previous comparison to generate a matching result with our method B. The parameters used for the two different techniques in Figure 7.12 are summarized below:

Houdini FEM solver on a small and a big sphere of radius 1, resp. 2

Tet embedding	**Physics**
max tet scale: 0.06	*stiffness multiplier*: 1000
min triangle scale: 0.03	*damping ratio*: 0.5
enlarge offset: 0.35 / 0.699	*mass density*: 1000
discretization max resolution: 1024	*shape/volume stiffness*: 2/20
	friction: 0.01
	impulse force: $(0, 0, -4)$ / $(0, 0, 40)$

Method-B

Global parameters	**Profile shape**
pseudo-stiffness τ: 0.2	$\mathbf{v}_2.x$: 0.0 / 0.10
deformation extent $\bar{\phi}$: 250 / 150	$\mathbf{v}_3.x$: 0.76 / 0.54
Relative stiffness $\bar{k}_{\text{big}} = 0.6$	
bulge $\gamma = 0.9$	

We indicate in blue the parameters for the big sphere and in red those of the smaller one. When they are the same for both objects, parameters are printed in black. Although our approach is designed to resolve local contacts, it manages to produce a very similar deformation. As can be seen in the video accompanying [BBG21], however, compared to the time-dependent simulation, our result lacks dynamic inertia effects.

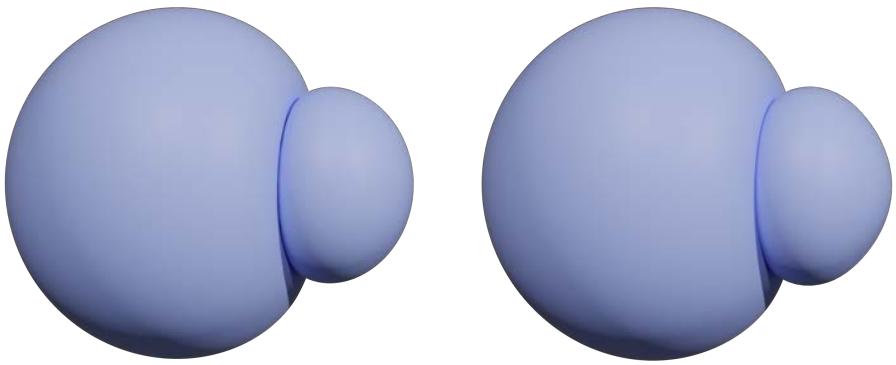


Figure 7.12.: Comparison of method B with a simulation. Left: FEM simulation in Houdini©SideFX. Right: method B.

2.2. Comparison to Implicit Skinning

In Figure 7.13 we compare our extended pipeline to a skinning technique handling contacts, the time-dependent *Implicit Skinning* technique of Vaillant et al. [VGB+14]. Both techniques properly resolve the contact and produces plausible bulges.

As already introduced in Chapter 2, this technique is based on the representation of the initial object by a set of implicit surfaces. It thus uses a volumetric representation of the mesh, each limb of a character being represented by a 3D scalar field stored as discrete grids. Compared to our surfacic representation of the input mesh, this has a much higher memory cost. Another advantage of our method is its time-independency. Indeed, whereas the 2013 version of Implicit Skinning was also fully time independent, to properly handle extreme character movements, the 2014 version of Implicit Skinning requires a dependency over the history of the animation.

The amount of work required to prepare the input object is also different. Our method requires only the identification of the different working and shared region (as illustrated in the inset of Figure 7.8). Those can be directly painted, or inferred from the skinning weights as a first guess. For the Implicit Skinning method, manual work is often required to adjust the location of the joints as well as the implicit surfaces of all limbs. Blending controllers also needs to be adjusted for each kind of articulation.

However, the Implicit Skinning method almost fully replaces the underlying geometric skinning, which is only used to guide the tracking of the resulting implicit surface. In contrast, our method is applied as a post-process of geometric skinning and therefore inherits all its limitations (including volume variations due to collapsing or bulging artifacts) but the self-intersections. In addition, our approach is sensitive to the quality of the deformation produced by the skinning. Nevertheless, it can handle contacts between any parts of the input mesh (or meshes), whereas Implicit Skinning is only able to response, with a reasonable cost, to collisions around rigid articulations of adjacent bones.

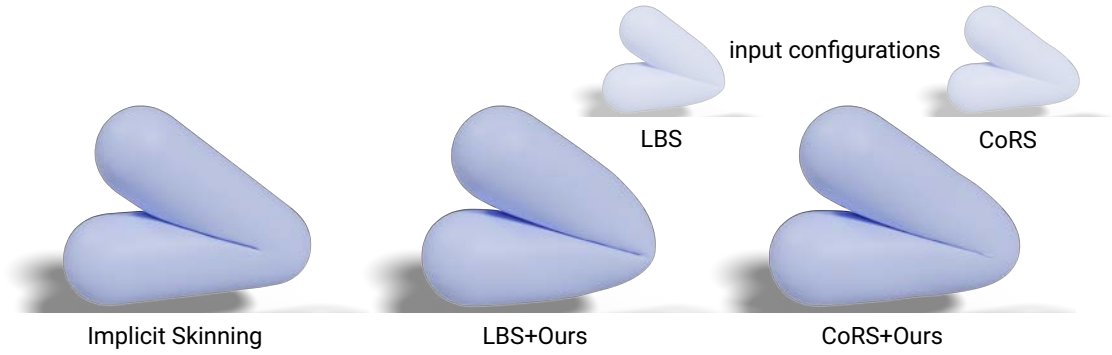


Figure 7.13.: Comparison of method B to Implicit Skinning [VGB+14] on the bending of a skinned capsule user either Liner Blend Skinning (LBS) or Center of Rotation Skinning (CoRS) as input to our deformer. Notice the volume preserving bulge produced by our deformer. [$\tau = -1$, $\bar{k} = 0.5$]

2.3. Comparison of methods A and B

Our two methods mostly differ from each other at two steps of the pipeline. First, the mapping between the two surfaces is computed differently. Method A uses a shared parametrization, which leads to distortion artifacts. In contrast, method B uses a unique mapping direction. Second, the methods for extracting the contact zone are also fundamentally different. In method A, the ball-testing approach uses a volumetric criterion that allows to take into account the local geometry of the input surfaces but is limited to the asymmetric configurations of a collision between an elastic and a rigid object. Method B alleviates this limitation using a surfacic geometrical symmetric criterion to handle elastic-elastic collisions. Contrary to the ball-testing method, this approach relies on a global threshold that does not allow to adapt to the local variations of the spatial relationships.

Figures 7.14 and Figure 7.15 compare our approaches in two cases for which both methods are applicable: a rigid finger pressing on the head of an octopus and rigid feet walking on an elastic plane. In simple configurations, such as the one illustrated in Figure 7.14, our two methods produce very similar results.

Mapping computation. As already noted in Section 2.1 of Chapter 4, method A struggles and eventually breaks when there is a too large distortion between the two intersected surfaces, such as when the whole foot goes below the ground. In contrast, the techniques to match the surfaces and extract the contact zone of method B are both agnostic to the actual intersection regions, which allows us to properly match the sole of the foot with its complete footprint on the ground.

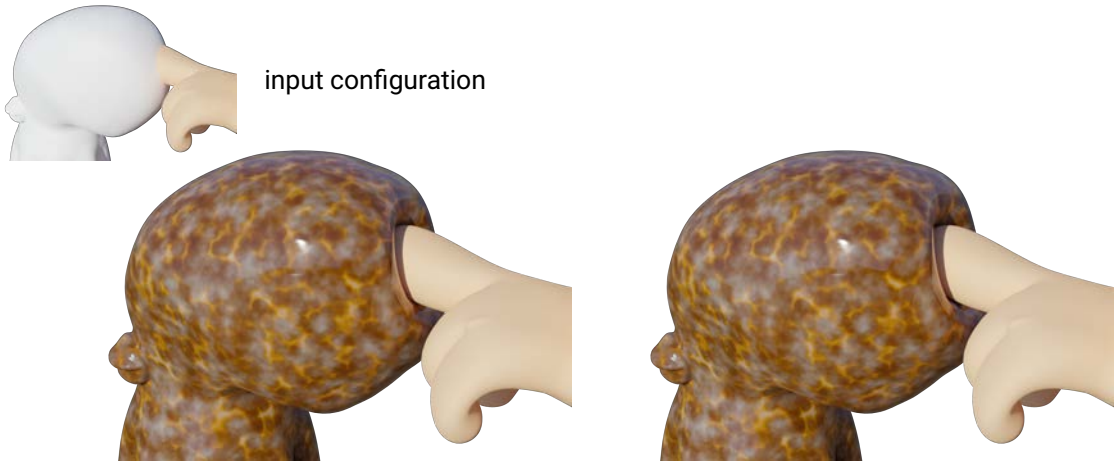


Figure 7.14.: Comparison of methods A and B. A rigid finger pressing on the skin of an octopus head using the approach of method A (Left) compared to method B (Right).[$r_s = 90$, $\tau = 0$, $\bar{\phi} = 300$, $\gamma_{\text{finger}} = 0$]

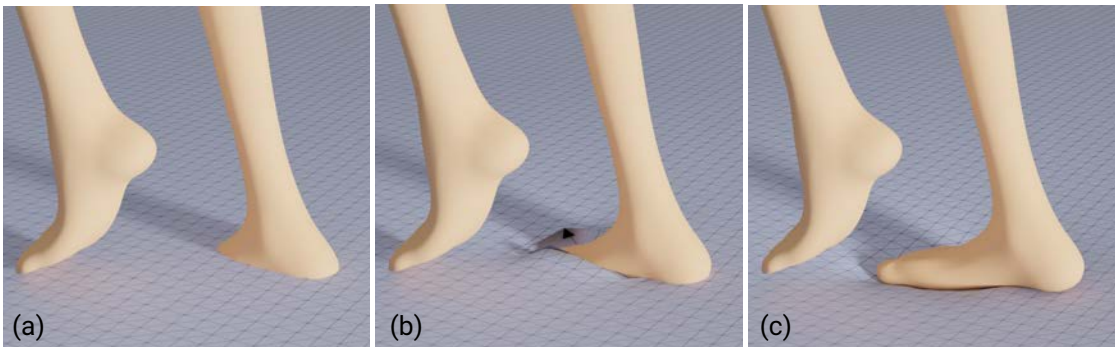
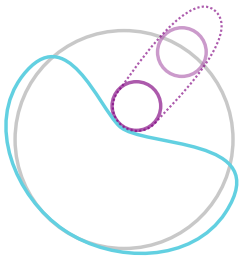


Figure 7.15.: Comparison of methods A and B. (a) A rigid foot colliding with an elastic plane using (b) the approach of method A compared to (c) method B.[$r_s = 5$, $\tau = 0$, $\bar{\phi} = 150$, $\gamma_{\text{plane}} = 0$]

Nevertheless, to ensure time-independence, both methods share the same limitation: they require a surface-to-surface intersection between the objects to produce a response. For instance, if a tiny rigid object (in purple, with position at a previous frame in light-purple) entirely crosses the elastic surface (in grey), then we could not generate any deformation response. For such extreme penetrations, even a volumetric collision detection would not be sufficient to figure out in which direction the elastic object should be squashed. A practical workaround to this problem would be to let the artist design a proxy shape bounding the rigid surface (dashed lines), hence creating an intersection with the elastic surface at the desired location.



Ball-testing strength. The ball testing approach of method A described in Section 4.1 of Chapter 4 allows us to handle challenging cases such as those presented in Figures 7.16 and 7.17: a rigid hand plunging in a soft cushion and a fully elastic hand pressed against a window. In those configurations, the hand corresponds to a unique working region producing a single contact region. As already mentioned in the discussion at the end of Chapter 4, the approach used by method B to determine the contact zone is not suitable here due to the global definition of the criterion ε_c. In these two examples, there is only one intersection zone, but multiple disconnected contact regions are merging and splitting during the animation. In the accompanying video of [BBGB20], it can be noted that this does not affect the temporal continuity of the deformation.

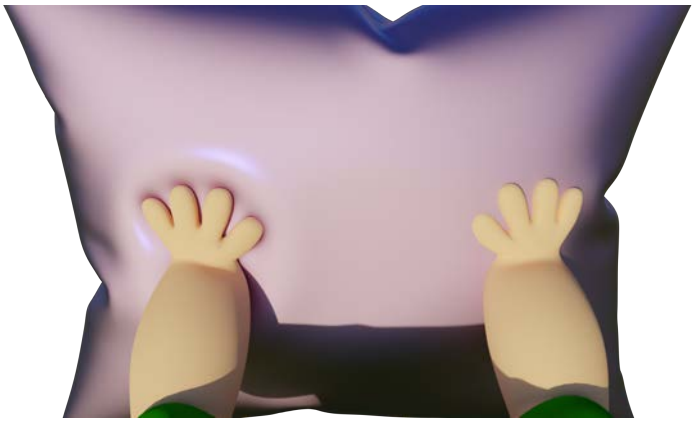


Figure 7.16.: Ball-testing strength. Two hands pressing on a pillow. The left hand is in simple intersection (the fingers are inside the pillow, while the palm has traversed it). The right hand has been deformed with method A to convey plausible contact and bulging effects.[Method A : $r_s = 140$, $\bar{\phi} = 300$, $\gamma_{\text{pillow}} = 0.8$]

Figure 7.17.: Ball-testing strength. Two hands pressing against a window. The right hand is in simple intersection (the fingers are inside the window, while the palm has traversed it). The left hand has been deformed with method A to convey plausible contact and bulging effects.[Method A : $r_s = 3$, $\bar{\phi} = 40$, $\gamma_{\text{fingers}} = 0.1$]

3. Artistic control

Artistic control is one of the main goals of our work. As already presented in the Chapter 4 and 5, multiple parameters are provided to the user to design the final deformation of the colliding surfaces and convey the desired physical characteristics of these objects:

- relative and apparent stiffness values (Chapter 4),
- extend of the deformation (Chapter 5, Section 1),
- volume repartition (Chapter 5, Section 4.2).

The effects of varying those user-controlled parameters have been demonstrated in the associated sections of the previous chapters. In practice, we noticed that the stiffness and the extend of the deformation should be generally correlated, but we preferred to leave them independent for full artistic freedom. Throughout this section we will present more advanced controls offered by our method.

3.1. Profile curve editing

First, additional controls are granted through the modification of the profile function $\mathcal{H}$ globally over the entire surface. In Figure 7.18, using method A, we consider a rigid plane and an elastic sphere of diameter 100. As shown in the middle in this figure, adjusting the abscissa of $\mathbf{v}_2$ and $\mathbf{v}_3$ permits to move the bulge radially. The profile function may also be modulated by an auxiliary procedural function, as demonstrated in Figure 7.18 (right) where a sinusoidal modulation is employed. Similar editing capabilities are offered by the method B, *i.e.*, profile-curve tweaking, as illustrated in Figure 7.19 where wrinkles have been easily generated by adding a sinusoid function to the profile-curve.

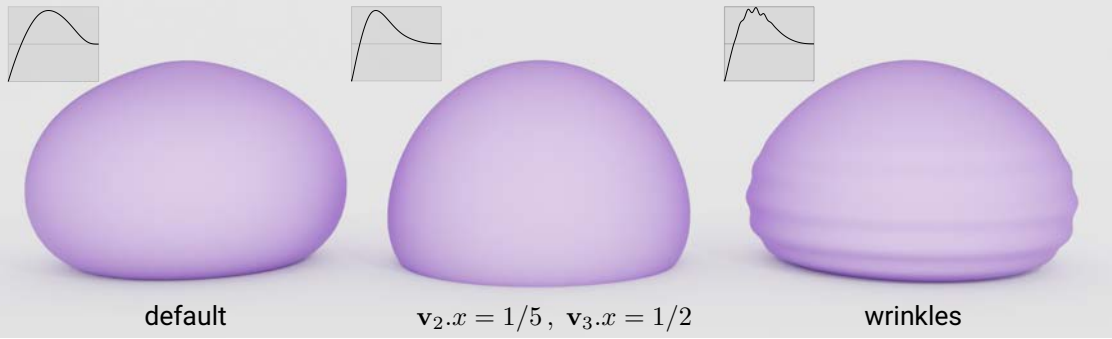


Figure 7.18.: Artistic profile curve variations. We visualize the effect of adjusting the abscissa of $\mathbf{v}_2$ and $\mathbf{v}_3$ (middle), or adding a sinusoidal function to mimic wrinkles (right). The respective profile curves for an average slope are shown in the insets. [Method A : $r_s = 25$, $\phi_{\max} = 60$ and $\gamma = 1$]

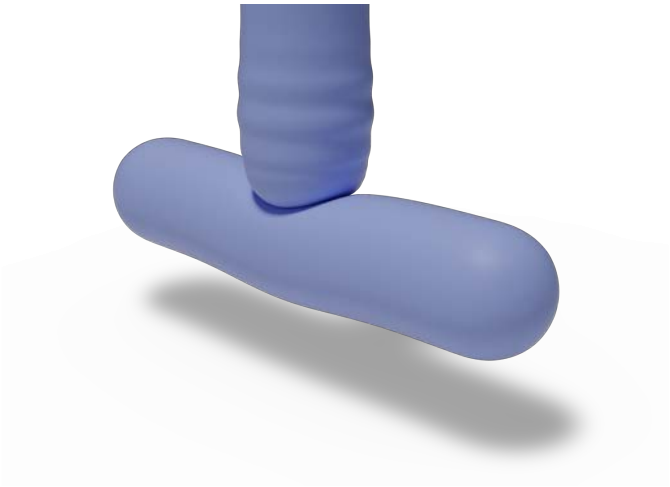


Figure 7.19.: Artistic profile curve variation. Wrinkles generated by adding a sinusoid to the profile curve of the vertical capsule. The horizontal capsule is deformed with an anisotropic bulge. [Method B : $\tau = 0$, $\bar{\phi} = 200$, $\bar{k} = 0.5$]

3.2. Manual parameter tuning

Most parameters used by our method can be defined automatically. Yet, for artistic purposes, we also give the possibility to the user to define them manually, and even potentially to keyframe them in time.

This approach is particularly relevant in the case of the mapping direction $\hat{\mathbf{d}}$ to compensate the lack of friction effects due to the time-independency. Indeed, to induce friction forces, we would need to know the movement of the objects prior to the current frame. To workaround this limitation, we can manually specify the mapping direction to give the impression of friction as illustrated in Figures 7.20 and 7.21. On the left of the first figure, the vertical mapping direction gives the impression that the finger is pressing vertically on the purple surface. In the middle one, resp. the right one, it looks like the finger is sliding on the deformable surface from left to right, resp. right to left.

Similar observations can be made on Figure 7.21. In the left figure, the vertical mapping direction gives the impression that the finger comes from the top of the octopus head and goes downward. In the right figure, given a horizontal mapping direction, we observe an asymetric bulge mimicing a frictional sliding effect. As expected, with the automatic mapping direction, shown in the middle figure, the finger just seems to press the skin at one particular point without any sliding.

Figure 7.22 illustrates the tricky example of a slightly soft hat pressing onto a softer egg-like head. This time it is the relative stiffness value that is manually animated to give the impression that a force is applied to put the hat on the egg-shaped surface until it reaches the point where the applied force becomes sufficient for the hat to fit the shape of the egg. To give this impression, the hat is initially almost fully rigid and the egg surface elastic, and the relative stiffness is progressively "transferred" from one object to the other during the animation. The cartoon effect is reinforced by slightly increasing the volume compensation factor ($\gamma = 1.2$) of the egg-shaped object to exaggerate the bulge effect. To better appreciate the effect, please watch the animated sequence corresponding to this scenario in the video associated to [BBG21].

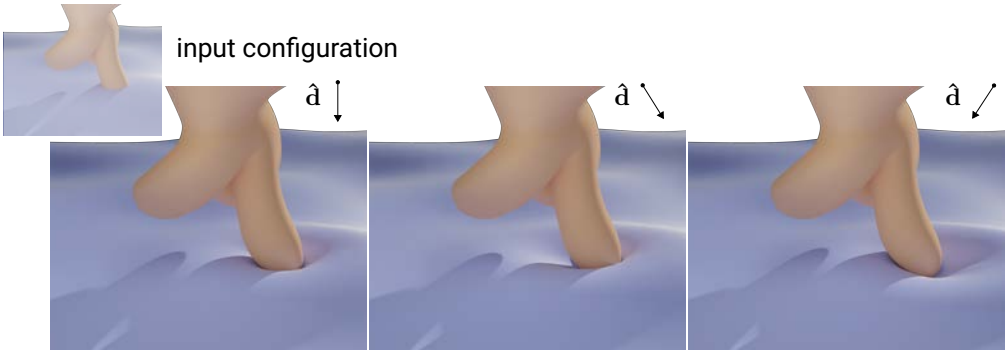


Figure 7.20.: Control over the mapping direction $\hat{\mathbf{d}}$. Left: using the automatic direction. Middle and Right: modifying the mapping direction permits to mimic friction effects. [Method B : $\tau = 0$, $\bar{\phi}_{\text{ground}} = 50$, $\bar{\phi}_{\text{finger}} = 20$, $\bar{k}_{\text{finger}} = 0.9$]

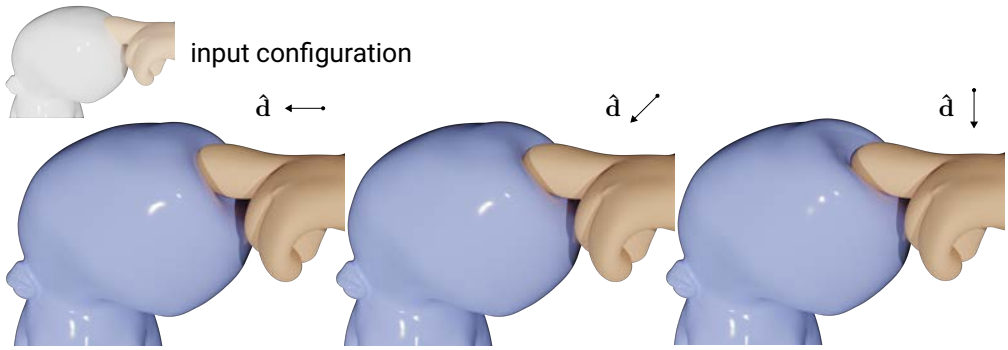


Figure 7.21.: Control over the mapping direction $\hat{\mathbf{d}}$. Middle: using the automatic direction. Left and Right: modifying the mapping direction permits to mimic friction effects. [Method B : $\tau = -0.3$, $\bar{\phi}_{\text{octopus}} = 30$, $\bar{\phi}_{\text{finger}} = 24$, $\bar{k}_{\text{finger}} = 0.9$]

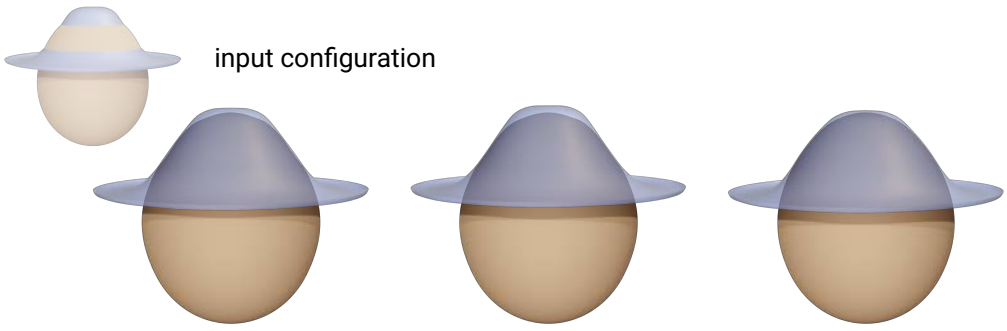


Figure 7.22.: Stiffness transition. From left to right, the relative stiffness is transferred from the hat to the egg. We see the *backface* of the hat surface. [Method B : $\tau = 0.2$, $\gamma_{\text{hat}} = 0.3$, $\gamma_{\text{egg}} = 1.2$]

3.3. Spatially-varying parameters

Instead of defining a constant parameter over the whole working region, we additionally offer the possibility to define some parameters locally using a painting tool.

We could imagine to paint any parameter used in our pipeline (e.g., the local stiffness of an object). Our current implementation is restricted to the parameters defined on the deformable region $\mathcal{D}$: the abscissa of $\mathbf{v}_2$ and/or $\mathbf{v}_3$ and the bulge repartition via the bulge map η.

In Section 5.2 of Chapter 5, we presented an automatic method to define this bulge map. However, we also provide the option for the user to paint this map directly on the initial mesh or interactively during the design session. To illustrate this functionality, in Figure 7.23, we consider a rigid sphere colliding with an elastic plane. This figure shows anisotropic *star-shaped* bulges obtained through a painted bulge map, giving the impression that the plane is in fact a peace of fabric. This is another way to produce wrinkles with our method in addition to the modification of the profile curve. Currently, the map is attached to the surface setting a constant value to each vertex for the duration of the whole animation. We could imagine attaching this map to the contact zone instead. For example in Figure 7.23 it could follow the sphere as it slides on the plane to retain this fabric feeling. Another possibility would be to animate the bulge map in time independently from the objects movement to create dynamic variations.

Finally, Figure 7.24 shows an example where we modify spatially the abscissa x_2 of the control point $\mathbf{v}_2 = (x_2, y_2)$ to vary the distance of the bulge to the contact zone. Let us denote by $x_{1,i}$, $x_{2,i}$ the actual abscissa of the control points $\mathbf{v}_1$, $\mathbf{v}_2$ for the vertex i. Here, the painted map encodes the relative position of the abscissa of the control point inside the interval $[0.2, x_3]$, with 0.2 corresponding to the maximum possible value of $x_{1,j}$ when the tangent vanishes at $\partial\mathcal{C}$. Slightly negative values of this painted map (depicted in blue in the bottom left figure) are tolerated since $x_{1,i}$ is rarely equals to its maximum possible value 0.2 in that configuration. However we must be careful with this inverted position of x_1 and x_2 if we want to produce plausible bulge, and more importantly to ensure that the profile curve remains a function of u (or x as both are equivalent here).

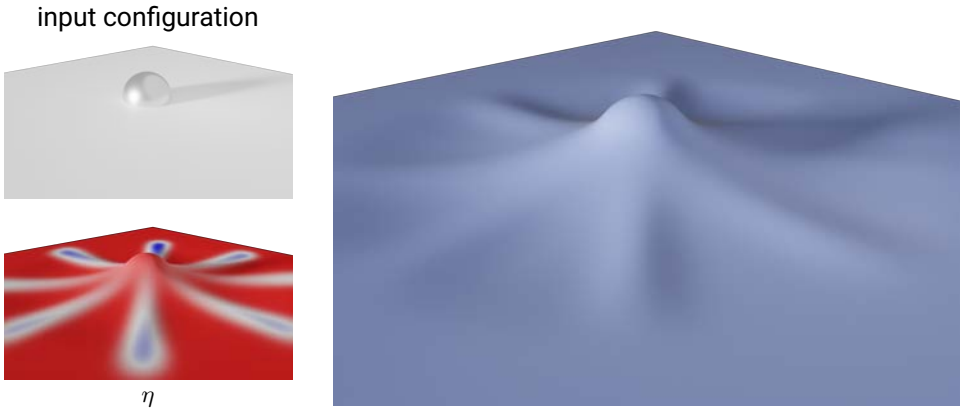


Figure 7.23.: Painted bulge map. A rigid sphere is pushing below a plane and a fabric-like effect is achieved through a painted bulge map η with values ranging from one (the default in red), to slightly negative values (in blue). Here we see the *backface* of the planar grid. [Method B : $\tau = 0$, $\bar{\phi} = 150$, $\gamma_{\text{plane}} = 0.4$, $\bar{k}_{\text{sphere}} = 1$]

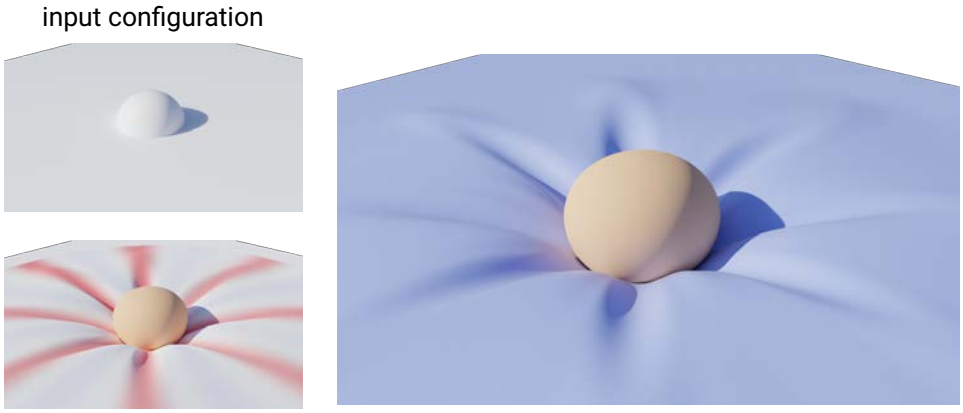


Figure 7.24.: Painted abscissa of $\mathbf{v}_2$. A sphere falling on a plane. Here the artistic bulge is obtained by varying the abscissa of $\mathbf{v}_2$ from its original automatic value. The painted map (bottom left) encodes the relative position of $\mathbf{v}_2$ in the interval between 0.2 (maximum abscissa of $\mathbf{v}_1$) and $\mathbf{v}_3$. This value can be inside the interval (red paint) or slightly below (blue paint). [Method B : $\tau = 0$, $\bar{\phi}_{\text{sphere}} = 60$, $\bar{\phi}_{\text{plane}} = 150$, $\gamma_{\text{sphere}} = 1$, $\gamma_{\text{plane}} = 2$, $\bar{k}_{\text{sphere}} = 0.8$]

4. Corner cases & limitations

In this section we present two corner cases that push our method to its limits (Section 4.1 and Section 4.2) as well as a study on the temporal continuity of our approach in the particular case of multiple disconnected components (Section 4.3).

4.1. Thin structures

Figure 7.25 shows a stress test for our method on a sphere-like shape made of thin tubular structures colliding with an equally stiff plane. As can be seen on the left, our method produces a convincing deformation even for the tubes that are deeply immersed in the plane. However, since our method does not use any volumetric structure, such as a tetrahedral mesh, the tubes gets significantly distorted when the collision goes deeper (as seen on the right) until they eventually collapse. For this specific example, a workaround would be to embed the tubular-sphere within a smooth genus-0 bounding triangular mesh (with bounded minimal curvature) onto which our algorithm would be applied, prior to transferring the deformation of this proxy to the detailed tubular mesh.

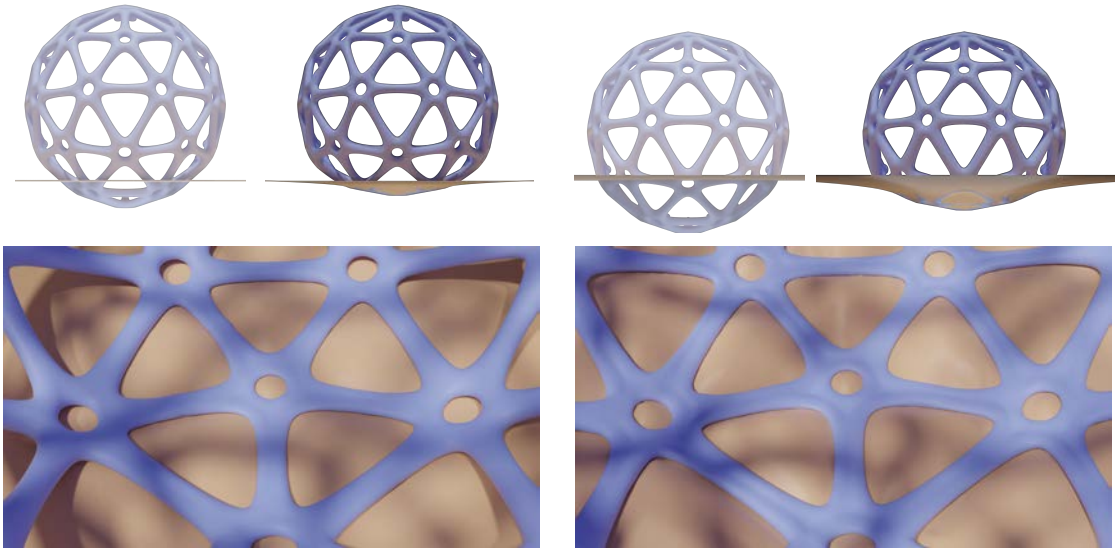


Figure 7.25.: Thin structures. An elastic tubular sphere collides with an elastic plane for two penetration depths (left and right columns). In both cases, our method produces plausible deformations of the overall shapes (first row), but for the deepest collision the tubes start to collapse (bottom right). [Method B : $\tau = 0$, $\bar{\phi}_{\text{plane}} = 20$, $\bar{\phi}_{\text{sphere}} = 10$, $\gamma_{\text{plane}} = 0$, $\gamma_{\text{sphere}} = 1$, $\bar{k}_{\text{sphere}} = 0.56$].

4.2. Folds

Figure 7.26 shows another stress test consisting in a sphere colliding with many bended thorns lying on a plane. This example is particularly challenging for the mapping algorithm of method B (Chapter 4, Section 2.2) as it creates many complex back/front

facing configurations along the mapping direction. Nevertheless, for most parts, the produced deformation remains plausible with some thorns being completely flattened, hence producing expected folds. This example reveals some limitations though.

First, since the spikes which are away from the contact zone are deformed along the surface normal, they will unnaturally inflate or shrink when the volume gets redistributed (top right inset in Figure 7.26). A simple workaround would be to guide the displacement direction along a smoothed normal field.

Second, we can see that some collisions between the spheres and the thorns are missed due to the global threshold used by the purely surfacic contact criterion of method B (Chapter 4, Section 4.2). This contrasts with method A which identifies the contact zone through a volumetric ball-testing procedure that better accounts for the spatial relationships and local variations (Chapter 4, Section 4.1). Designing a new heuristic combining the best of both approaches makes an interesting future work.

Third, this example recalls that both methods can miss or anticipate local collision events when a new collision occurs nearby a bulge or a depression produced by a previous collision. The many colliding thin thorns exacerbate this issue to the point of producing some flickering during the animation, as shown in the accompanying video of [BBG21].

For less challenging configurations, such as the examples of Figure 7.5 and 7.6, those theoretical discontinuities are not noticeable. This is thanks to both the locality of our fields and the localization term of the anisotropic bulging weights. For the simpler case of two successive collision events, the second one occurring on a previous prominent bulge or depression, a practical workaround would be to apply sequentially our deformer twice, starting with the deepest collision, instead of processing all the components at once. Designing a more general solution to this problem is left for future work.

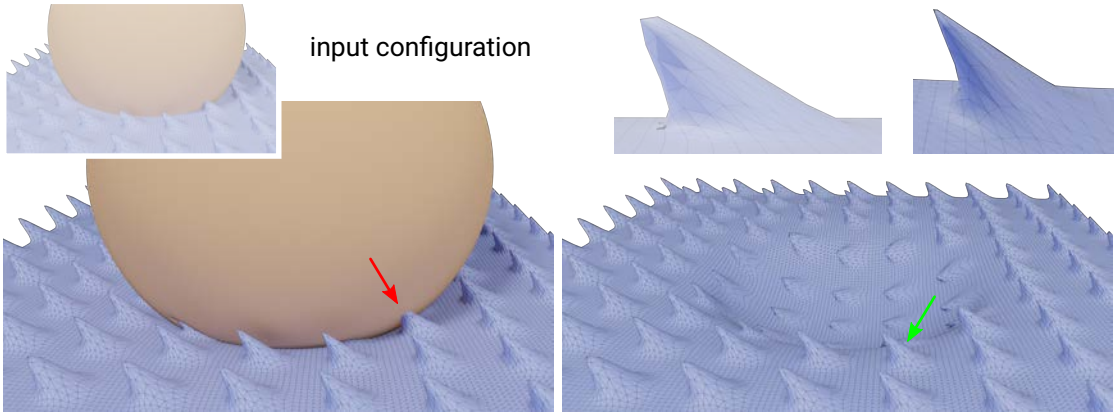


Figure 7.26.: Folds. An elastic sphere collides with many bended elastic thorns. Top-right: close-ups of the thorn highlighted by the green arrow before and after deformation. Bottom-right: same as bottom left but with the sphere hidden to reveal the expected folds. The red-arrow highlights a missed collision. [Method B : $\tau = -0.8$, $\bar{\phi}_{\text{sphere}} = 20$, $\bar{\phi}_{\text{plane}} = 4$, $\gamma_{\text{sphere}} = 1$, $\gamma_{\text{plane}} = 0$, $\bar{k}_{\text{sphere}} = 0.8$]

4.3. Temporal continuity

Even though we designed our pipeline to ensure temporal continuity throughout the animation, we strived to assess more precisely to which extent method B respects this constraint in the case of multiple disconnected components.

Figure 7.27 shows a "unit" test to objectively and quantitatively evaluate the sensitivity of our pipeline to sudden changes of the deformation region. It consists of a plane and two spheres, one of which is already in collision with the plane. In this experiment, we vary the vertical position of the second colliding sphere, and we measure the displacement speed of the plane vertices. More precisely, we tracked the elevation of all vertices, and reported in Figure 7.27 its temporal variation for the vertex $\mathbf{p}$ exhibiting the largest "jump". It is located in the deformation area of both spheres. Since the initial surface is a plane and the mapping direction is vertical, this quantity is proportional to the signed amplitude of the displacement of $\mathbf{p}$.

As a reference, we consider the same configuration without the static sphere. In this case, the variation of the amplitude is continuous over the whole animation (blue curve in Figure 7.27). However, once we add back the static sphere, a leap is noticeable when the contact of the moving sphere is first detected (green and red curves). This discontinuity is due to two factors.

The first one is the time independence of our method: as discuss earlier, since we do not take into account the previous frame and thus previous deformations, our pipeline is applied on the initial surfaces at each frame. If an object collides with an already deformed surface in a bulged region, it will not be detected. If we look at the graph of Figure 7.27, the displacement of the point $\mathbf{p}$ before the contact is positive, which means that it is located on the bulge produced by the collision of the static sphere.

The second factor is related to our definition of the deformed regions. They are defined around the contact zones until a user-controlled distance is reached. As a consequence, these regions just appear when a contact is detected with their associated 1D parametrization. When a first contact is detected, the displacement amplitude diffused over the region is null, which guarantees the temporal continuity observed on the reference configuration. But when a second contact occurs, the amplitudes resulting from the two contacts are blended together over the deformed region by the diffusion process. The resulting amplitude field is thus greater than zero, which accentuates the magnitude of the discontinuity. This issue is emphasized when the amplitude and slope at the boundary of the contact zones greatly vary. To alleviate this problem we need to reinforce the locality of these two scalar fields to reduce the influence on the deformation of a contact as we move away from it.

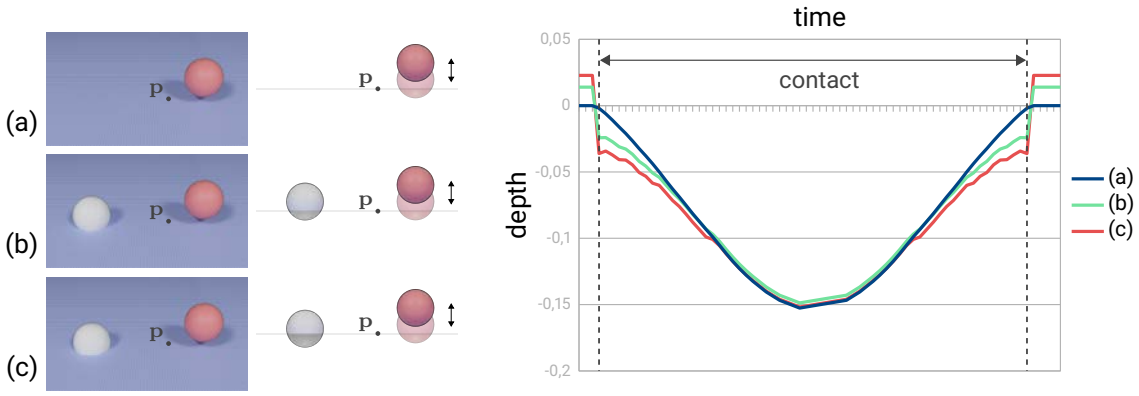


Figure 7.27.: Temporal continuity of method B. We study the elevation of the point **p** on an elastic plane in collision with a rigid sphere. The three studied configurations are presented on the left. The position of the white sphere is fixed whereas the red one has a vertical movement from top to bottom and back to top, as shown in the side views of the second column. On the right, the elevation of the point **p** is plot for the three configurations over the course of the animation.

5. Performance

In this section we will present the performance of our implementation of both methods A and B. We measured the performance of our prototypes on a single core of an Intel i7-4790K CPU. Table 7.1 reports breakdown timings for method A, whereas average breakdown timings for method B are reported for several examples in Table 7.2.

On average the processing timing of method A ranges from 6ms to 730ms depending on the working region complexity (Table 7.3). It ranges from 6ms to 175ms for method B, with a maximum mesh size slightly smaller for this test set. To evaluate the scaling of our method with respect to the size of the input meshes, we deliberately included over-tessellated meshes such as the jelly of Figure 7.10, the planar ground of Figure 7.15, or the simple shapes of Figure 7.22. Overall, our implementation of method A achieves interactive performance for an elastic working region whose complexity ranges from $2k$ and up to $4k$ vertices, which is very promising as significantly better timings could be obtained through basic code refactoring. The new algorithms proposed in method B are less computationally demanding than those of method A, and we put some more effort in their optimization. As a result, our implementation achieves interactive rates when the working regions of both objects count from $10k$ and up to $20k$ vertices. Here the two involved meshes are deformed and considered all along our pipeline. It is worth noting that these computation time are highly dependent from the portion of the working regions $\mathcal{W}_l$ contained in the deformable ones $\mathcal{D}_l$. When comparing the performance of method A and B, please keep in mind that method A only handles rigid/elastic configurations. The deformation is therefore calculated only on the elastic object and most of the timings only concerns the corresponding mesh. On the contrary, the timing of method B has been measured on elastic/elastic configurations except for the walk

cycle (Figure 7.15) and the egg-shape surface with the hat (Figure 7.22) configuration. The deformation of the two involved meshes are thus included in the resulting measures.

For both methods, the extent of the working region provides a tradeoff between quality of the deformation and performance. On the one hand, the larger this user-defined region is, the closer to a pseudo-geodesic distance the parametrization will be. On the other hand, the cost of the computations performed over this whole region (i.e., intersection and parametrization steps) will increase. For instance, for method A on the comb-bunny test case, those two steps represent more than 75% of the computation time, since the deformable region only account for 12% of the working region vertices.

Putting low-level implementation details aside, the major bottlenecks are expected to be the spatial searches of Chapter 4 (intersection, mapping) and the multiple sparse matrix factorizations. We discuss those two aspects in the two following sections.

5.1. Spatial searches

The formers are embarrassingly parallel and could thus greatly benefit from a multi-threaded or even a GPU implementation. Also note that intersection computations could be sped up using state-of-the-art techniques such as joint BVH traversal and batch intersection tests instead of testing each segment at a time in random order as currently done in our prototype. This step could also benefit from the collision detection literature (e.g., [Eri04, TKH+05]).

Regarding spatial searches, the ball-testing/sliding step (Chapter 4, Section 4.1) is by far the most time consuming part of method A, even though it is only carried out on a small subset of edges. It represents an important bottlenecks of this approach, in addition to its limitation to the asymmetric configurations. We envision great optimization opportunities of the ball-sliding step by devising a dedicated AABB-tree traversal, exploiting the fact that one of the ball extremities is empty since it was already accepted at the ball testing stage. In comparison, the contact zone extraction of method B is one of the cheapest step of the entire pipeline thanks to its simplicity.

5.2. Matrix factorizations

If we now take a look at the multiple sparse matrix factorizations needed by our two methods, we can notice that they represent the majority of the timing in both cases. Method A involves 6 sparse linear solves at every frame, which currently requires 5 matrix factorizations,because the different Poisson problems are carried out on slightly different domains while involving different least-squares linear constraints. Only the diffusion of the amplitude and slope fields shares the exact same matrix.

Unsurprisingly, the computation time of method B is dominated by the ray-triangle intersections but also the three matrix factorizations, two for the heat-method and one for the diffusion of the slope and tangent parameters. As previously mentioned in Chapter 5 and contrary to method A, if the treated configuration presents only one mapping direction, no diffusion is needed for the direction field definition. In the case of multiple mapping directions, instead of a complex matrix factorization, only a real matrix

factorization is required and we even can use the same one as the amplitude and slope diffusion.

Moreover in method B, to optimize the 1D-parametrization step (Section 1 of Chapter 5) we sped up the two matrix factorizations needed for the heat-method by unifying them through SIMD vectorizaton. This is possible because the two matrices have the exact same structure. Implementation-wise, we took advantage of Eigen's template genericity to feed it with a custom scalar type holding a pair of scalars within SSE single precision floating point registers. In practice this allows us to perform the two factorization at the cost of about 1.3× the cost of a single one.

Overall, in this second approach, the factorizations themselves represent about 25% to 60% of the overall computation time depending on the ratio between the working and deformable regions. We believe that their cost could be significantly cut down by adapting prefactorization techniques [HDA17, HA18]. This would mean pre-factorizing the standard Laplacian matrix over the whole region of interest on the elastic objects only once, and then localize and adjust the problems through fast, structure-preserving, rank updates. Herholz et al. reported up to 10× speedups with such a technique.

Table 7.1.: Runtime statistics of method A for a pair of rigid and elastic meshes with a number of vertices (# vert) inside the deformable region $\mathcal{D}$; average computation time over an animation broken down in terms of detecting the regions in intersection (i.), building the mapping between them (m.), extracting the contact zone (c.), computing the deformation region parametrization (p.), diffusing the directions of deformation (di.) and the guiding fields (g.), and deforming the surface according to the profile with $\gamma = 1$ (de.), in percentage of the average total computation time (total).

scenes	$\mathcal{D}$	**relative time (%)**							**total**
	(# vert)	**i.**	**m.**	**c.**	**p.**	**di.**	**g.**	**de.**	**time (ms)**
caps - caps	2608	19.3	4.4	15.4	22.0	19.0	7.8	12.1	50.8
caps - v. caps	1120	20.8	2.7	14.4	23.4	14.8	9.0	14.9	18.9
caps - caps	1913	16.7	1.3	9.4	32.4	14.3	9.0	16.8	29.7
v. caps - caps	875	18.4	1.9	12.4	39.5	7.4	5.0	15.5	23.4
plane - sphere	568	17.9	3.4	17.5	20.3	16.8	9.4	14.8	9.93
sphere - plane	1456	12.5	2.2	15.6	21.6	17.7	11.6	18.7	16.9
s. sphere - b. sphere	9995	4.1	0.6	21.0	26.5	25.4	13.7	8.7	233.1
ball - torso	407	22.1	4.5	12.1	19.9	13.0	10.5	17.9	6.2
comb - bunny	4413	10.1	0.6	8.5	65.2	4.1	1.9	9.7	378.4
finger - squid	13144	4.6	0.5	10.2	53.5	19.2	5.7	6.4	646.8
hand - pillow	18612	5.2	0.8	32.0	31.2	19.4	5.0	6.3	728.5
window - hand	1064	48.2	6.7	7.1	18.8	12.7	1.8	4.8	69.1

Table 7.2.: Runtime statistics of method B for pairs of working regions with various number of vertices; average computation time over an animation broken down in terms of detecting the regions in intersection (i.), building the mapping between them (m.), extracting the contact zone (c.), computing the deformation region parametrization (p.), computing the directions of deformation (di.), diffusing the guiding fields (g.), and deforming the surface according to the profile (de.), in percentage of the per frame average total computation time (average). The column (s.) reports the percentage of time spent in matrix factorizations.

scenes	$\mathcal{D}_1 + \mathcal{D}_2$ (# vert)	**relative time (%)**								**average time (ms)**
		i.	**m.**	**c.**	**p.**	**di.**	**g.**	**de.**	**s.**	
fingers - ball	194 + 1479	11	9	1	26	4	30	18	33	5.6
fingers - palm	616 + 1202	23	9	1	23	3	21	20	26	9
skinned finger	1163 + 981	28	9	1	26	2	20	14	27	13.6
mouth, fig. 7.7	1320 + 1050	25	13	3	24	1	18	15	26	9.8
bunny - jelly	4182 + 4259	12	9	2	43	1	18	16	41	101.8
walk, fig. 7.15	0 + 11682	5	4	0	30	1	49	11	61	76.8
hat - egg	10137 + 0	41	12	2	30	1	10	5	33	173.4

Table 7.3.: Number of vertices (# vert) inside the working region $\mathcal{W}$ of the meshes used in the test cases.

mesh	caps	vert. caps	plane	sphere	small sphere	big sphere	torso	ball
# vert	3028	1465	1521	639	635	10183	529	521

mesh	bunny	comb	squid	finger	hand	pillow	window	hand
# vert	35542	1831	29871	1209	2285	26234	22862	2314

mesh	fingers	ball	fingers	palm	phalanx 1	phalanx 2	up. lip	low. lip
# vert	1064	1561	2664	2184	1967	2951	1825	1500

mesh	bunny	jelly	feet	plane	hat	egg
# vert	14408	5002	740	11717	12092	10242

8

Conclusion

Throughout this thesis, we strived to develop a deformation technique for elastic objects in collision that provides interactive feedbacks to the user with parameters that can be intuitively controlled. We were not seeking for physical realism, but rather for plausible results that are produced quickly, intuitively and with real-time feedbacks for the computer animation domain.

In this manuscript, we have presented a new deformation technique for the resolution of local elastic-elastic contacts while producing plausible bulge effects and providing extended artistic control. It does not aim at generating large, global deformations that should already be handled by skinning, blend shapes, or cage-based deformers. For example, it should not be used to resolve the collision of the whole, two-sided ear of the Stanford Bunny with its back. By relying entirely on geometric operations, we successfully managed to achieve instant feedback with no time dependency for a seamless integration in the animation pipeline. Thanks to a careful design of each steps to work on continuous boundary curves, temporal continuity over the course of an animation is ensured without introducing any temporal dependencies. Finally, by using a combination of an automatic 1D parametrization and a 2D profile curve, we manage to produce volume preserving plausible bulges. This association also allows artistic bulge control, from no to full volume preservation and even exaggeration, as well as a wide range of deformation details introduced using art-directed tools.

We also proposed an extension of our pipeline to handle local self-intersections, hence enabling geometric skinning with contact handling, bulge control, and other artistically driven effects.

Even though we did not rely on physics laws as the classical physical simulation methods, the plausibility of our final deformations is strengthened by an anisotropic spreading of the volume-preserving bulge. Moreover, in addition to be able to convey information about the physical properties of objects using few parameters, our approach allows the artist to design cartoony effects using volume exaggeration, but also volume repartition using one of the numerous artistic controls that we proposed: global or local profile curve manipulation, manual or spatially-varying parameters, etc.

This type of approach opens new avenues for the deformation of elastic objects in contact in the context of computer animation: editing may be performed non linearly,

similar deformations can be replicated in different contexts without requiring tedious manual adjustments, and many artistically driven effects can be easily accomplished by tweaking the profile curve.

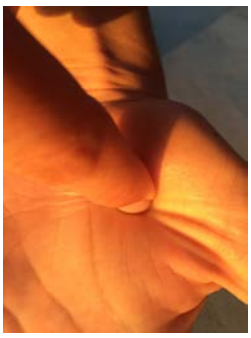
Perspectives

Throughout this manuscript we have already presented and discussed many technical opportunities for improving some parts of our pipeline. In this section, we will discuss more general open questions and perspectives.

User study. First, as we developed a tool for artists, it would have been nice to have their feedback on the usability and usefulness of this work through a user study. This implies, however, that one should have access to both amateur and professional users of such tools. In the same vein, it would be interesting to test our method on real production rigs. Almost all the results of this thesis have been rigged and animated by non professional artists. Moreover, comparisons with more complex and professionally designed simulations or captured deformations could also be helpful to validate the plausibility of the deformations produced by our method.

Link contact and diffusion criterions. One observation that we did not take into account in this approach is the correlation between the stiffness of a material and the extent of the deformable region. In our current implementation, this size is set arbitrarily by the user. Nevertheless, we have the intuition that the stiffer the material is, the less it can be deformed locally and therefore the deformation needs to extend further from the contact to fully compensate the volume lost in the collision region. As the threshold used in the definition of the contact zones is also related to the stiffness of the material as discussed in Section 4 of Chapter 4, we believe that the two quantities determining the regions of interest of our method should be correlated. Linking them together automatically would ease the work of the artist.

From our 1D to a 2D parametrization. Moreover, apart from the anisotropic bulge repartition introduced in Section 5.2 of Chapter 5, the bulge deformation defined by our method is roughly the same all around the contact regions resulting in deformations that do not always look very natural.



In the photo on the right, we can see wrinkles starting from the contact zones. These details are due to structures located underneath the skin (bones, muscles, ligaments, etc.). To give another example, if you press your thumb on the inside of your wrist, a bulge will only appear in the direction transversal to your forearm. This bulging is, in part, the result of the lateral displacement of tendons. To offer a large range of plausible deformations to the user, we cannot ignore such configurations and restrict ourselves to

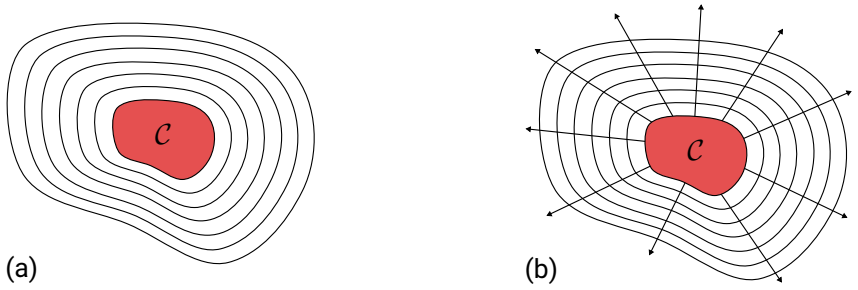
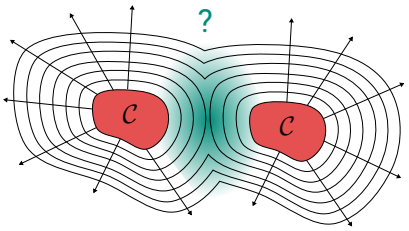


Figure 8.1.: 2D parametrization of the deformable region. (a) In the current method, the region around the contact zone is equipped with a 1D parametrization to instantiate the profile curve. (b) A 2D parametrization of this zone would allow the user to modulate this profile around the contact.

ideal ones assuming a homogeneous inner body structure. Therefore, it would be interesting to extend our approach to produce an artistically-controllable anisotropic deformation around the contact region to be able to reproduce such deformations. With our painting tool, we can already produce such deformation effects but at the cost of a manual painting of the vertices for every collision. Moreover, what if one of the colliding objects slides over the other? In order to design a more automated solution, one possibility would be to dynamically define a 2D parameterization of the deformed region instead of a simple radial 1D parametrization as in our current pipeline. This 1D parametrization corresponds to the approximate geodesic distance of any point on the deformed mesh to the contact region (Figure 8.1 (a)). We can thus only know on which iso-value curve of the distance to the contact area is located a vertex of the mesh. With a 2D parameterization (Figure 8.1 (b)), we would also know precisely to which geodesic any vertex belongs.



This 2D parametrization would thus allow us to propose a second profile curve to control the shape of the bulge anisotropically around the contact region. We have already considered this problem during this thesis and the challenging part is to define a coherent 2D parametrization when there are more than one contact region as illustrated in the inset figure. What are the values in the 2D parametrization expected in the green region?

Previous work already explored this type of parametrization on meshes, for example using local exponential maps [HA19]. However such parametrizations are defined around a single point of interest. In our case, it needs to be defined around one or even multiple closed regions. One possible research direction could be to adapt and extend those methods to our problem. Interactivity is a mandatory condition in our application domain and it seems rather difficult to reach real-time performances with this kind of techniques. Therefore, it would be worthwhile to explore faster approximations that should still produce a local 2D parameterization which is as smooth as possible. If we manage to compute such a map efficiently, it will also permit to replace some of the costly diffusions in our original method and thus to improve its computation time.

In the same vein, in our approach, we assume that the stiffness is constant over a whole working region, which is quite limiting regarding the same idea of non-homogeneous inner body structures. Indeed, pressing on a bone or a relaxed muscle should not produce the same deformation. What would happen if we press the same muscle near a rigid bone? What would the resulting bulge of the skin over a bone look like in comparison with an adjacent region without any rigid inner structure? It would be interesting to investigate the possibility to handle the deformation of a region with a spatially varying stiffness, potentially painted as a pre-process by the user.

Skin elasticity. To improve the plausibility of our deformations, even in extreme cartoony configurations, we could also take into account more physical properties of the skin, such as its elasticity in addition to its stiffness already considered. Indeed, our displacements are performed for each vertex along a direction field without taking into account the stretching that may be undergone by the skin with the application of the deformation. This stretching can be further worsened by the artistic exaggeration of the volume preservation. This would imply allowing tangential movements of the vertices to limit these stretches and to find a direct way to integrate them in our method.

Procedural details. Moreover, when the skin is deformed, the dilatation or contraction of the meso-geometry of the surface and the micro-structure of the material might significantly change the reflectance properties of the surface once integrated within illumination models. For instance, skin folds, including wrinkles, might appear or disappear during the deformation. Some parts may also change from red to white and vice-versa due to variations of blood concentration. A low scale deformation issued from the skinning could thus be complemented with such high-frequency details and effects. A combination of procedural approaches with a multi-resolution appearance model may be worth investigating. Another simpler solution to add details with our current implementation could be to paint directly on the surface underlying hidden details that would be revealed during the collision process, for example wrinkles in front of a sliding finger on the forearm or at the beginning of a skin pinch.

Self-intersecting skinning method. Since it is applied as a post-process, the output of our pipeline highly depends on the smooth skinning method used and their resulting mesh deformations. Current skinning techniques and their associated rigging parameters are designed to prevent self-intersection by making the deformations unrealistically soft. To exploit our deformer at its full potential, it would be relevant to investigate novel skinning and rigging techniques focusing on the generation of nice deformations on the exterior visible part of the joint, while deliberately producing self intersections on the interior. Moreover our volume-preserving deformation does not correct the initial volume lost by the skinning method and, even though we are able to compute this loss, our deformer is not tailored to correct it. In order to have a full preservation of the volume from the rest pose of an articulated model, the solution would be to either design a volume preserving skinning method, or before the application of our deformer, to apply

a volume correction using a method like the ones of [vFTS08] or [RHC09].

Other animation principles. Finally, in this research, we only considered (half of) one of the twelve animation principles presented by Disney: *the squash* (from "Squash and stretch"). It would be challenging but exciting to investigate others of these principles with the same philosophy of providing the artist with intuitive automatic but artistically-driven tools to produce the associated effects. Recently, Zang et al. [ZBLJ20] and Rohmer et al. [RTK+21] proposed two methods to add secondary motion effects over an animated skinned character. The former used simulations producing motions orthogonal to the skinning deformation, while the latter proposed a geometric solution relying on the skeleton movement. Both approaches produce less robotic, more lively animations without requiring a heavy and hardly art-directable full-simulation applied after the animation and rigging stage.

All these approaches pave the road to new ways of creating tomorrow's animations.

Appendix

A

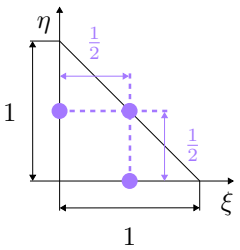
Gaussian quadrature for triangular domain

In practice, most integrals either cannot be evaluated analytically, or their evaluation is very lengthy and tedious. Consequently, for simplicity, numerical integration methods are often preferred to get approximate computation of an integral using numerical techniques. A wide range of methods exist for numerical integration, also called quadrature, and can be expressed as: "Given a sampling of m points of some function f over a domain Ω, find an approximation of $\int_\Omega f$." A quadrature form is :

$$\int_\Omega f(\mathbf{x})d\mathbf{x} \approx \sum_i \omega_i f(\mathbf{x}_i)\,,$$

where Ω is the designated integration region, and f is an integrand defined on Ω. The points $\mathbf{x}_i$ are called quadrature nodes, and ω_i are the quadrature weights.

The Gaussian quadrature technique is an optimal method as its goal is to obtain the best numerical estimate of an integral by picking optimal abscissas $\mathbf{x}_i$ at which to evaluate the function $f(\mathbf{x})$. It produces the most accurate approximation possible. If we considered m quadrature nodes, Gaussian quadrature fits all polynomials up to degree $2m-1$ exactly. In that case, the optimal abscissas correspond to the roots of the m^{th} Legendre polynomial. In our method, we deal with gaussian quadrature on triangular faces (i.e., Ω is a 2D domain). As the formulas are independent of the triangle shape, the idea is to transform the triangular element to the standard triangle T illustrated in the inset figure and then apply the quadrature:



$$\iint_T f(\xi,\eta)d\xi d\eta \approx \frac{1}{2}\sum_i^m \omega_i f(\xi_i,\eta_i)\,, \qquad \text{(A.1)}$$

where m is the number of quadrature points, (ξ_i,η_i) are quadrature nodes located inside the standard triangle and ω_i are weights (normalized with respect to the triangle area). We want to choose nodes (ξ_i,η_i) and weights ω_i in Equation A.1 so that the quadrature is as accurate as possible in some sense.

For our application we use a quadrature of degree 2 with three quadrature nodes that are the mid-points of the sides of the triangle as illustrated in the inset figure. Other possibilities may be used ; for example, the trisection points of the medians that are not the centroid could be used as the points of evaluation. By definition, the quadrature should be accurate for $f(\xi, \eta) = 1$, ξ, η, ξ^2, $\xi\eta$, and η^2, leading to the following nodes and weights:

$$(\xi_1, \eta_1) = \left(0, \frac{1}{2}\right), (\xi_2, \eta_2) = \left(\frac{1}{2}, 0\right), (\xi_3, \eta_3) = \left(\frac{1}{2}, \frac{1}{2}\right), \omega_1 = \omega_2 = \omega_3 = \frac{1}{3}$$

The Gaussian quadrature of degree 2 for the standard triangle T with three nodes can be expressed as :

$$\iint_T f(\xi, \eta)\, d\xi d\eta \approx \frac{1}{6}\left[f\left(0, \frac{1}{2}\right) + f\left(\frac{1}{2}, 0\right) + f\left(\frac{1}{2}, \frac{1}{2}\right)\right]$$

B

Laplacian and contraints

With all the diffusions necessary throughout our method, the Laplace operator Δ is a core element of our pipeline. In this appendix we will make a quick overview of this operator definition and characteristics. Then, in a second section, we will discuss the boundary conditions and the linear constraints required for our application.

1. Laplace operator

The Laplace operator, or Laplacian, is a well known second order differential operator which plays a central role in many domains of physics (elasticity, quantum mechanics, ...) and computer science (smoothing, geometric deformations, ...). Indeed, many physical models and signal processing tasks reduce to Laplacian-based differential equations. The Laplace-Beltrami operator generalizes the definition of the ordinary Laplacian from Euclidean space to functions defined in curved domains Ω, and is usually denoted as Δ_Ω. As we work on surfaces, we will use this version of the operator but, for simplicity, we will keep using the Laplacian for the Laplace-Beltrami operator and refer to it with the same notation Δ. In general, this operator is define as the divergence or the gradient of a function u, i.e., $\Delta u = \nabla \cdot u$.

The intuition behind the Laplacian of a signal is that it gives the deviation of a value at a point $\mathbf{x}_i$ from a local average defined by a small sphere around $\mathbf{x}_i$. In physics, for example, the heat equation $\frac{d}{dt}u = \Delta u$ describes how the temperature u is diffused in an environment over time. If we seek the equilibrium of this quantity in an environment Ω with fixed values at its boundaries, we then consider the steady-state heat equation which boils down to the Laplace equation:

$$\Delta u|_\Omega = 0, \text{ with } \quad u|_{\partial\Omega} = u_0$$

The Laplacian comes with some interesting properties. A first basic property is that constant functions are in the kernel of the Laplace equation. A second property is that the Laplacian of a function is invariant to rigid motion and isometry. This second property is very useful for us for performance reasons: we can compute this operator on our input mesh once and there is no need to compute it at each frame if the considered

object is transformed by a rigid motion. Finally, the Laplacian is an elliptic differential operator. Therefore, if we define an energy and minimize it in term of the Laplacian, we get a convex energy.

The solutions of the Laplace equation are called *harmonic* functions. They are also minimizers of the Dirichlet energy which measures the smoothness of a function over a domain:

$$E_D(u) = \frac{1}{2}\int_\Omega \|\nabla u(\mathbf{x})\|^2 d\mathbf{x}, \text{ with } \quad u|_{\partial\Omega} = u_0,$$

where u_0 is a given function defined on $\partial\Omega$.

This is not the only interesting property of harmonic functions. Indeed, in addition, they meet the following principles:

- the *mean value property*: the value of a harmonic function at a point is equal to its average value over any spheres or balls centered at that point.

- the *maximum principal* : maximum and minimum values of a harmonic function are located on the boundaries of the domain of definition. No extrema can be found on interior points.

Discretization The discrete Laplacian is commonplace in many algorithms, for instance in physical simulation, machine learning and geometry processing. For such applications, when working on polygonal meshes, the discrete version of the previously defined operator is used. It is based on the assumption that meshes can be interpreted as piecewise linear approximation of smooth surfaces. Different approaches have been proposed to compute the approximation of the differential properties of the underlying surface directly from the mesh data. Therefore, many definitions exist to express the discrete Laplacian. Yet all of them lead to the following matrix formulation of the Laplace equation as a linear system:

$$\mathrm{L}u = 0,$$

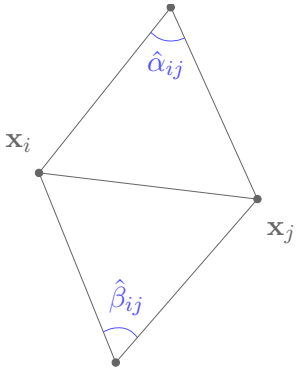
where the matrix L is called the *Laplacian* matrix or also the *stiffness matrix*.

For example, this discretization can be derived using a finite element method. Without going into too much details, the idea is to find a linear combination of basis functions φ to approximate our function u with $\tilde{u} = \sum_i v_i\varphi_i$, where $v_i \in \mathbb{R}$ are the nodal values. Projecting the residual onto each basis functions and reducing it to its *weak form* yields: $L_{ij} =< \nabla\varphi_i, \nabla\varphi_j >$. For further details, the reader is invited to take a look at one of the numerous resources on this subject, such as, for instance, the **Discrete Differential Geometry** course of Keenan Crane [KCS13].

In computer graphics, the most widely used discretization are piecewise linear functions, with nodal values v_i attached at the mesh vertices. In this case, we obtain the well

known cotangent formula:

$$
\begin{aligned}
L_{ij} &= \frac{\cot \hat{\alpha}_{ij} + \cot \hat{\beta}_{ij}}{2} \text{ , if } j \in N_i, \\
L_{ii} &= -\sum_{j \in N_i} L_{ij} \\
L_{ij} &= 0 \text{ , otherwise,}
\end{aligned}
$$



where N_i corresponds to the set of neighbors of $\mathbf{x}_i$, $\hat{\alpha}_{ij}$ and $\hat{\beta}_{ij}$ are the angles opposite to the edge between the vertices i and j, as illustrated in the inset.

Note that when triangles have obtuse angles, the cotangent coefficients may become negative, which can lead to flipped triangles in some applications. A simple workaround consists in subdividing the input mesh [Riv84], but other discretizations alleviate this limitation [Flo03, BS07].

Translating this definition in the matrix form $\mathrm{L}u = 0$ with L the Laplacian matrix, we get:

$$
\begin{aligned}
L_{ij} &= \frac{\cot \hat{\alpha}_{ij} + \cot \hat{\beta}_{ij}}{2} \text{ , if } i \neq j, \\
L_{ii} &= -\sum_{\mathbf{x}_j \in N_i} L_{ij}
\end{aligned}
$$

One major advantage of the discrete Laplacian operator matrix is that it is very sparse. Its memory cost can thus be highly reduced using sparse data structure and the system of equations can be efficiently solved with sparse solvers.

2. Boundary conditions and linear constraints

The boundary conditions and constraints play a crucial role in determining the final solution. Many kinds of boundary conditions exist, but here we will just mentioned the two main ones: Dirichlet and Neumann conditions.

Boundary conditions Up to now, we only describe the Laplace equation with Dirichlet boundary conditions. Basically it consists in fixing the value of the function u at the boundary of the domain Ω.

Alternatively Neumann conditions allow to fix the derivatives of that function at the boundary of Ω. The Laplace equation then becomes:

$$
\Delta u|_\Omega = 0, \text{ with } \left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = g \,,
$$

where n is the outward pointing normal vector along the boundary and g a given function defined on $\partial\Omega$. We must be careful when choosing the boundary conditions: a partial derivative equation may not have solution for a given boundary condition.

Linear constraints In several steps of our pipeline, the boundary conditions of the Laplace equation are not located on vertices of the mesh but on a set of points lying on its edges. One possibility would have been to cut the considered triangles dynamically at each frame and use classical Dirichlet boundary conditions. However it would have raised several issues. First, the topology of the mesh changing during the animation may expose us to temporal continuity issues. Second, there is also the high risk of producing degenerate triangles that would lead to poorly conditioned cotangent Laplacian matrices.

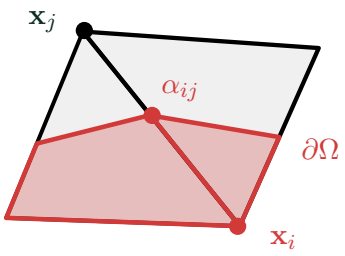
Instead, our solution is to define the boundary conditions as linear constraints of the form $(1-\alpha_{ij})\,u_i + \alpha_{ij}\,u_j = b_{ij}$, with $\alpha_{ij} \in [0,1]$ for all edges $i-j$ crossed by the boundary of the considered domain Ω. Since we discretize the Laplace equation on a triangular mesh equipped with linear elements, we cannot guarantee in general that both the linear constraints and Laplace equation can be satisfied exactly. In practice, we thus turn them into soft constraints by solving them in the least-squares sense. In matrix form, this yields:

$$(\mathrm{L} + \beta\,\mathrm{C}^\top\mathrm{C})\,v = \beta\,\mathrm{C}^\top \cdot \mathbf{b}\ ,$$

where L is the cotangent Laplacian matrix, C is the matrix holding the left-hand-side of the linear constraints, and β is a weight balancing the diffusion with boundary terms. Observing that the norm of the rows of L are expected to be roughly 10 times larger than the ones of $\mathrm{C}^\top\mathrm{C}$, and that L is negative semi-definite, we set $\beta = -10^2$ to give slightly more weights to the linear constraints.

Moreover, to avoid temporal discontinuities when the boundary of the domain crosses a vertex, we are especially careful when assembling the Laplacian matrix. The integrals $L_{ij} =< \nabla\varphi_i, \nabla\varphi_j >$ of the gradients of the linear basis-function φ_i must be computed over the true working region Ω. This is typically the case of the heat-diffusion and Poisson problem when computing our 1D parametrization where the boundary of Ω corresponds to $\partial\mathcal{C}$.

In practice, this means that the three cotangent terms associated with a triangle face overlapping the boundary of our domain need to be weighted by the ratio between its area after being cut by $\partial\Omega$ (inset figure, grey region) and its full area. Specifically we set:



$$L_{ij} = \frac{w_{ij}\cot\hat{\alpha}_{ij} + w_{ji}\cot\hat{\beta}_{ij}}{2}\ ,\ i \neq j$$

where w_{ij}, resp. w_{ji}, is the area ratio associated to the triangle corresponding to the angle $\hat{\alpha}_{ij}$,resp. $\hat{\beta}_{ij}$.

These adjustments prove to be crucial to ensure the temporal coherence of the diffused fields, and thus of the final deformation, despite the time-independence constraint of our method.

C

Volume constraint equation

In this appendix we detail the equation used to ensure the volume preservation constraint at the end of our pipeline, as described in Section 4.2 of Chapter 5.

Let us recall Equation 5.5 that computes the displacement vector between a point $\mathbf{p}_i$ of the input mesh and its final position $\mathbf{p}'_i$ on the deformed surface:

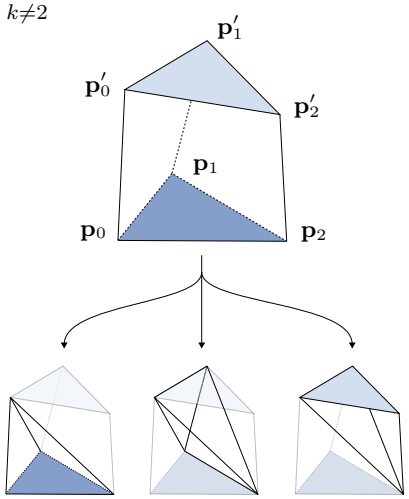
$$\mathbf{p}'_i - \mathbf{p}_i = \mathbf{d}_i \left(h_v \, N_2^3(t_i) + \sum_{\substack{k=0 \\ k \neq 2}}^{5} y_k \, N_k^3(t_i) \right),$$

where $t_i = f_1^{-1}(u_i)$, y_k is the ordinate of the control point $\mathbf{v}_k$ and N_k^3 corresponds to the k^{th} cubic B-spline basis function. We must determine h_v such that the original volume is preserved at the end of the deformation.

Reorganizing the terms and isolating the unknown, we express $\mathbf{p}'_i$ as:

$$\mathbf{p}'_i = h_v \, \mathbf{l}_i + \mathbf{c}_i, + \mathbf{p}_i \quad \text{with} \quad \begin{cases} \mathbf{l}_i = \mathbf{d}_i \, N_2^3(t_i) \\ \mathbf{c}_i = \mathbf{d}_i \sum_{k \neq 2} y_k \, N_k^3(t_i) \end{cases}$$

To measure the contribution of one face to the volume displacement, we must consider the volume of the triangular prism whose bases are the initial face and its corresponding one at the end of the deformation. We can compute the volume of this prism as the sum of the volume of three tetrahedra as illustrated in the inset.



As a reminder, the volume of a tetrahedron ABCD can be expressed as :

$$V_{ABCD} = \frac{1}{6} (\overrightarrow{AB} \times \overrightarrow{AC}) \cdot \overrightarrow{AD} = \det(\overrightarrow{AB} \, \overrightarrow{AC} \, \overrightarrow{AD})$$

Therefore, the contribution to the volume displacement of the face f is:

$$
\begin{aligned}
V_f &= V_{(\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}'_0)} + V_{(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}'_0,\mathbf{p}'_1)} + V_{(\mathbf{p}_2,\mathbf{p}'_0,\mathbf{p}'_1,\mathbf{p}'_2)} \\
&= \frac{1}{6}\,(\det(\mathbf{p}_1-\mathbf{p}_0, \mathbf{p}_2-\mathbf{p}_0, \mathbf{p}'_0-\mathbf{p}_0) \\
&\quad + \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{p}'_0-\mathbf{p}_1, \mathbf{p}'_1-\mathbf{p}_1) \\
&\quad + \det(\mathbf{p}'_0-\mathbf{p}_2, \mathbf{p}'_1-\mathbf{p}_2, \mathbf{p}'_2-\mathbf{p}_2)) \\
&= \frac{1}{6}\,(\det(\mathbf{p}_1-\mathbf{p}_0, \mathbf{p}_2-\mathbf{p}_0, h_v\,\mathbf{l}_0+\mathbf{c}_0) \\
&\quad + \det(\mathbf{p}_2-\mathbf{p}_1, h_v\,\mathbf{l}_0+\mathbf{c}_0+\mathbf{p}_0-\mathbf{p}_1, h_v\,\mathbf{l}_1+\mathbf{c}_1) \\
&\quad + \det(h_v\,\mathbf{l}_0+\mathbf{c}_0+\mathbf{p}_0-\mathbf{p}_2, h_v\,\mathbf{l}_1+\mathbf{c}_1+\mathbf{p}_1-\mathbf{p}_2, h_v\,\mathbf{l}_2+\mathbf{c}_2)) \\
&= \frac{1}{6}\,(\quad\ \det(\mathbf{p}_1-\mathbf{p}_0, \mathbf{p}_2-\mathbf{p}_0, \mathbf{c}_0) \\
&\quad + h_v\ \ \det(\mathbf{p}_1-\mathbf{p}_0, \mathbf{p}_2-\mathbf{p}_0, \mathbf{l}_0) \\
&\quad +\quad\ \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{p}_0-\mathbf{p}_1+\mathbf{c}_0, \mathbf{c}_1) \\
&\quad + h_v\,(\det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{l}_0, \mathbf{c}_1) + \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{p}_0-\mathbf{p}_1+\mathbf{c}_0, \mathbf{l}_1)) \\
&\quad + h_v^2\ \ \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{l}_0, \mathbf{l}_1) \\
&\quad +\quad\ \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{c}_2) \\
&\quad + h_v\,(\det(\mathbf{l}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{c}_2) + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{l}_1, \mathbf{c}_2) \\
&\qquad + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{l}_2)) \\
&\quad + h_v^2\,(\det(\mathbf{l}_0, \mathbf{l}_1, \mathbf{c}_2) + \det(\mathbf{l}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{l}_2) + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{l}_1, \mathbf{l}_2)) \\
&\quad + h_v^3\ \ \det(\mathbf{l}_0, \mathbf{l}_1, \mathbf{l}_2)
\end{aligned}
$$

It can be factorized as a degree three polynomial in h_v:

$$V_f = \frac{1}{6}\,(h_v^3\,a_f + h_v^2\,b_f + h_v\,c_f + d_f)\,,$$

with :

$$
\begin{aligned}
a_f &= \det(\mathbf{l}_0, \mathbf{l}_1, \mathbf{l}_2)\,, \\
b_f &= \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{l}_0, \mathbf{l}_1) + \det(\mathbf{l}_0, \mathbf{l}_1, \mathbf{c}_2) + \det(\mathbf{l}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{l}_2) \\
&\quad + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{l}_1, \mathbf{l}_2)\,, \\
c_f &= \det(\mathbf{p}_1-\mathbf{p}_0, \mathbf{p}_2-\mathbf{p}_0, \mathbf{l}_0) + \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{l}_0, \mathbf{c}_1) + \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{p}_0-\mathbf{p}_1+\mathbf{c}_0, \mathbf{l}_1) \\
&\quad + \det(\mathbf{l}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{c}_2) + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{l}_1, \mathbf{c}_2) \\
&\quad + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{l}_2)\,, \\
d_f &= \det(\mathbf{p}_1-\mathbf{p}_0, \mathbf{p}_2-\mathbf{p}_0, \mathbf{c}_0) + \det(\mathbf{p}_2-\mathbf{p}_1, \mathbf{p}_0-\mathbf{p}_1+\mathbf{c}_0, \mathbf{c}_1) \\
&\quad + \det(\mathbf{p}_0-\mathbf{p}_2+\mathbf{c}_0, \mathbf{p}_1-\mathbf{p}_2+\mathbf{c}_1, \mathbf{c}_2)\,.
\end{aligned}
$$

Summing up the contribution of all the faces of the input mesh, we eventually obtain the following cubic equation in h_v:

$$h_v^3\, a + h_v^2\, b + h_v\, c + d = 0\,,$$

with :

$$a = \sum_{f \in F} a_f\,,\; b = \sum_{f \in F} b_f\,,\; c = \sum_{f \in F} c_f\,,\; d = \sum_{f \in F} d_f\,,$$

where F corresponds to the set of faces of the input mesh.

We solve this equation finding the three roots corresponding to the eigenvalues of the 3×3 companion matrix associated to this equation and define h_v as the smallest positive one.

D

Résumé en français

1. Introduction

Les mondes virtuels sont des univers parallèles qui nous permettent de nous échapper de notre vie quotidienne. Bien qu'imaginaires, ils doivent, pour être convaincants, se référer au monde réel que nous connaissons par le biais d'indices visuels. Dans le contexte de l'animation traditionnelle dessinée à la main, les animateurs Frank Thomas et Ollie Johnston ont théorisé dans leur livre de 1981 "The Illusion of Life : Disney Animation" [TJ81], douze principes de base de l'animation afin de produire l'illusion que les personnages de dessins animés sont conformes aux lois fondamentales de la physique. Ces principes sont toujours pertinents aujourd'hui pour les personnages virtuels animés par ordinateur [Las87].

De nombreux objets de notre environnement quotidien présentent des déformations élastiques lorsqu'ils entrent en contact avec un autre objet : un chat marchant sur un oreiller, une main appuyant sur une fenêtre, ou même nos joues écrasées par notre grand-mère chaque fois que nous la voyons. Ces objets ont tendance à s'écraser à l'intérieur de la zone de contact et à se gonfler lorsque leur volume est redistribué à l'extérieur de celle-ci. La représentation de ce type de matériau dans un monde virtuel doit préserver au mieux ces propriétés de l'apparence et du mouvement de l'objet pour être convaincante. Ces effets d'écrasement et de gonflement sont essentiels pour communiquer des déformations plausibles. Ils sont particulièrement importants dans l'animation de personnages, par exemple lorsque les artistes cherchent à traduire les actions d'un personnage sur l'environnement ou sur un autre personnage (saisir, pousser, presser, etc.). À titre d'illustration, la Figure D.1 présente divers exemples de contacts élastiques de la peau des personnages dans des productions industrielles.

La création d'animations 3D est un processus complexe qui nécessite différentes étapes et compétences artistiques. Une organisation et une structure solides sont donc nécessaires pour mener à bien un projet tel que la production d'un film d'animation. En particulier, une fois tous les personnages et les éléments de décors conçus par les modeleurs, il est temps de leur donner vie en suivant le storyboard donné par le réalisateur. Pour pouvoir être animés, ils doivent être équipés de différents systèmes connectés entre

Figure D.1.: Différents types de contacts observés dans les films d'animation. En haut à gauche et en bas à droite : la collision entre un objet rigide et un objet élastique est illustrée par les mains de Coco sur la fenêtre et le ventre de BayMax écrasé par l'armure. En haut à droite : les auto-collisions sont également présentes notamment sur la peau des personnages articulés, ici sur l'articulation du bras. En bas à gauche : la collision entre objets élastiques en général est illustrée par les joues de Bao et de la femme.

eux : le système de mouvement, le système de contrôle et le système de déformation. Ces trois éléments sont généralement regroupés sous un seul terme : le *rigging*, qui se réduit souvent à l'animation des squelettes. Grâce à ces outils, l'animateur peut contrôler les modèles numériques comme un marionnettiste anime ses marionnettes.

Cette thèse se concentre sur le système de déformation et en particulier sur la déformation d'objets élastiques lors d'une collision en animation 3D. Les particularités de ce domaine d'application imposent plusieurs exigences dans la conception des outils utilisés.

Déformations : les besoins des artistes

Comme évoqué précédemment, l'animation produite dans les films d'animation doit être crédible. Cependant, en tant que reflet de l'imagination du réalisateur et des artistes, elle est très imagée, tant au niveau des poses que des déformations. Elle retranscrit non seulement des événements, mais également des émotions. Malheureusement pour l'artiste, les déformations attendues s'écartent souvent de notre monde réel et dépendent en général

du style d'animation et de l'esthétique recherchée. De plus, les animateurs conduisent le spectateur à l'essence d'une situation, par exemple en utilisant l'exagération. Pour la collision entre des objets élastiques, l'écrasement traduit les propriétés physiques des objets (masse, rigidité, etc.), mais un écrasement exagéré peut également être utilisé pour amplifier l'effet de la collision. Pour toutes ces raisons, nous pouvons supposer que la précision physique n'est pas une exigence dans l'animation 3D. Les simulations physiques peuvent créer des mouvements réalistes mais elles ne sont pas adaptées à la stylisation des mouvements car contraintes par les lois de la physique. Dans ce travail, nous viserons donc plutôt la *plausibilité* pour préserver cette liberté artistique caractéristique de l'animation cartoons. Néanmoins, comme mentionné précédemment, certains indices visuels doivent être conservés pendant les déformations pour que celles-ci restent plausibles et crédibles. Par exemple, lorsqu'une balle frappe un sol dur, à moins qu'elle ne soit trouée, elle doit conserver à peu près son volume et il en va de même pour la peau du personnage. Une autre exigence concernant la collision est la production d'un contact propre. Lorsque deux objets entrent en collision, ils doivent être clairement en contact à un moment donné.

Enfin, afin de pouvoir reproduire l'animation souhaitée par les réalisateurs, le rigging des personnages et le rigging en général tentent de fournir aux animateurs le plus de flexibilité et de contrôle possible. La production d'un film d'animation entier exige une grande quantité de travail, car ils ne reposent pas sur des plans filmés comme les longs métrages à effets spéciaux. Le *contrôle artistique* combiné à l'*interactivité* pour ne pas ralentir le travail des artistes sont donc des caractéristiques essentielles à prendre en compte dans le développement d'un outil de déformation dans ce domaine.

Déformations : les solutions techniques existantes

Les objets élastiques sont particulièrement difficiles à animer par un artiste, surtout lorsqu'ils entrent en collision les uns avec les autres, car reproduire leur comportement d'écrasement et d'étirement implique de créer manuellement des déformations plausibles à la fois dans l'espace et dans le temps (par exemple en utilisant des déformeurs de type cage [NS13]). L'approche classique pour résoudre ce problème repose sur la physique des objets élastiques, soit en simulant explicitement leurs déformations (e.g., [NMK+06]), soit en s'appuyant sur des approximations plausibles mais plus rapides (e.g., [MCKM15]). L'avantage évident de ces approches est leur réalisme, à condition que les artistes parviennent à trouver les paramètres physiques qui produisent le comportement recherché. En pratique, une telle approche ne fournit à l'artiste qu'un contrôle indirect du comportement élastique et nécessite une expertise, ainsi que de nombreux essais-erreurs fastidieux, même pour un expert en simulation. De plus, elle ne permet pas de produire facilement l'exagération caractéristiques des animations cartoons. En outre, la principale limite des simulations dans un contexte d'animation est leur dépendance au temps : la simulation doit être exécutée depuis la première image jusqu'à l'image courante à chaque fois que l'artiste modifie un paramètre ou l'animation d'entrée. Par conséquent, en raison de cette dépendance temporelle, les simulations physiques doivent être exécutées après les étapes de rigging et d'animation, ce qui empêche l'édition non-linéaire de la scène 3D.

À l'autre extrémité du spectre méthodologique, les solutions alternatives sont les approches manuelles, entièrement contrôlées par l'artiste, comme celles basées sur les Blend Shapes, les déformations *free-form* [SP86] ou en espace des poses [LCF00]. Leur principal avantage est leur simplicité : elles fournissent un retour instantané aux artistes, qui sont ensuite chargés de produire des déformations convaincantes. En pratique, les effets de gonflement restent rares en production car la tâche de sculpter les déformations et de les animer à la main demande un temps considérable, même pour les artistes accomplis. Pire encore, chaque déformation est spécifique à la forme des objets et à leurs configuration de collision, et ne peut donc pas être réutilisée dans des situations différentes.

Pour l'animation de personnages, une classe intermédiaire de solutions [MZS+11, GMS14] produit des déformations indépendantes du temps à l'aide de simulations quasi-statiques — même si les déformations dues à des contacts distants dépendent toujours du chemin parcouru jusqu'à l'état de collision. Ces approches peuvent générer des déformations plus générales que les déformeurs purement géométriques, mais elles sont beaucoup plus exigeantes en termes de calcul en raison de leur nature itérative. De plus, ces méthodes offrent peu de contrôle à l'artiste et ne peuvent pas produire les déformations exagérées typiques des cartoons car elles sont basées sur les lois de la physiques. Les approches de modélisation géométrique traitant les collisions (e.g., [LB19]) peuvent être appliquées dans ce contexte, mais elles nécessitent une optimisation coûteuse et manquent de contrôles artistiques de la déformation.

L'objectif principal de cette thèse est de développer un outil de déformation qui assiste l'artiste en gérant les contacts et les effets de gonflement de manière contrôlable. Comme mentionné précédemment, une intégration transparente dans les pipeline d'animation nécessite : (1) que l'outil fournisse un retour instantané à l'artiste ; et (2) que les déformations soient indépendantes du temps pour permettre une édition non-linéaire de l'animation. Pour des effets de gonflement plausibles, il est également souhaitable que la méthode préserve le volume dans une certaine mesure, même si des contrôles artistiques doivent également être possibles pour explorer des réponses exagérées.

Seules quelques techniques de déformation fournissent des solutions à la fois interactives et indépendantes du temps. Cependant, elles présentent également d'importantes limitations pratiques. Les déformeurs procéduraux, tels que celui proposé par [Wan15] ou ceux intégrés dans les logiciels industriels comme le plugin *iCollide* d'Autodesk Maya© ou le *déformeurs de collision* de Cinema4D© offrent des déformations instantanées avec contrôle du profil de la déformation mais sont limitées au cas asymétrique d'une collision entre un objet rigide et un objet élastique. Ils se concentrent sur l'interactivité et le contrôle artistique et ignorent la préservation du volume, ce qui donne souvent lieu à des déformations peu naturelles. Enfin, d'autres méthodes fournissent des approximations (assez coûteuses) de la préservation du volume et sont limitées à des configurations spécifiques, telles que les articulations voisines d'un personnage articulé [VBG+13], ou la déformation d'une cage qui approxime la surface élastique [APHS11].

2. Notre méthode

Dans cette thèse nous avons voulu proposer une méthode automatique de déformation *locale* des surfaces élastiques suite à une collision. Nous n'avons pas traités les déformations globales des objets qui peuvent affecter les structures de contrôle comme les cages ou les squelettes par exemple et donc impacter le travail des animateurs. Notre objectif général est de proposer un outil indépendant du temps permettant une édition non-linéaire de l'animation, chaque image devant être traitée instantanément, indépendamment de la précédente ou de la suivante pour s'intégrer au mieux à l'étape d'animation du pipeline. Nous n'avons pas pour objectif de proposer une méthode de déformation physiquement réaliste qui n'offre qu'une très faible liberté artistique. Néanmoins, nous voulons produire des déformations plausibles en garantissant une préservation du volume pouvant être exagérée ou au contraire atténuée selon la volonté de l'artiste. Dans la même optique, nous voulons offrir un contrôle artistique suffisant pour permettre à l'utilisateur de réaliser les déformations qu'il a imaginées en amont ou souhaitées par le réalisateur.

Pour atteindre ces objectifs, nous avons développé une méthode qui traite, à chaque image indépendamment, la collision d'objets élastiques représentés par des maillages triangulaires. Notre méthode est basée sur l'observation qu'une collision entre deux objets est caractérisée par une déformation sans intersection qui peut être décomposée en deux parties : une *zone de contact*, où les deux surfaces resteront en contact, et un *gonflement*, au voisinage de cette zone de contact, résultant du volume déplacé lors de la résolution de l'intersection entre les objets en collision. Cette décomposition nous permet de simplifier le problème et de séparer notre méthode en deux grandes étapes. Dans un premier temps, nous déterminons la surface de contact entre les deux objets pour résoudre la collision. Dans un second temps, nous calculons une déformation autour de cette région de contact pour donner l'impression du déplacement du volume dû à l'impact. Notre pipeline est illustré dans son intégralité par la Figure D.2 et détaillé dans les sections suivantes.

Il peut être appliqué avec quelques étapes supplémentaires au cas des personnages articulés. Les deux parties de l'objet considéré peuvent alors être adjacentes sur une même surface, par exemple dans la configuration d'un bras plié avec une collision située dans le creux du coude comme dans la Figure D.1.

2.1. Définition du contact

Comme illustré en Figure D.2(a), notre pipeline prend en entrée un ensemble de régions de travail $\mathcal{W}_l$, chaque $\mathcal{W}_l$ englobant les régions déformables attendues.

Détection de collision. À chaque image, la première étape pour la gestion de contact consiste à détecter de manière robuste l'intersection entre les deux surfaces considérées. Pour ce faire, nous exécutons un algorithme de détection de collision arête/face entre les deux régions de travail. Il en résulte un ensemble de régions d'intersection $\mathcal{I}_l$ ainsi que les points d'intersection exacts entre les deux objets en collision représentés par des croix rouges dans la Figure D.2(b).

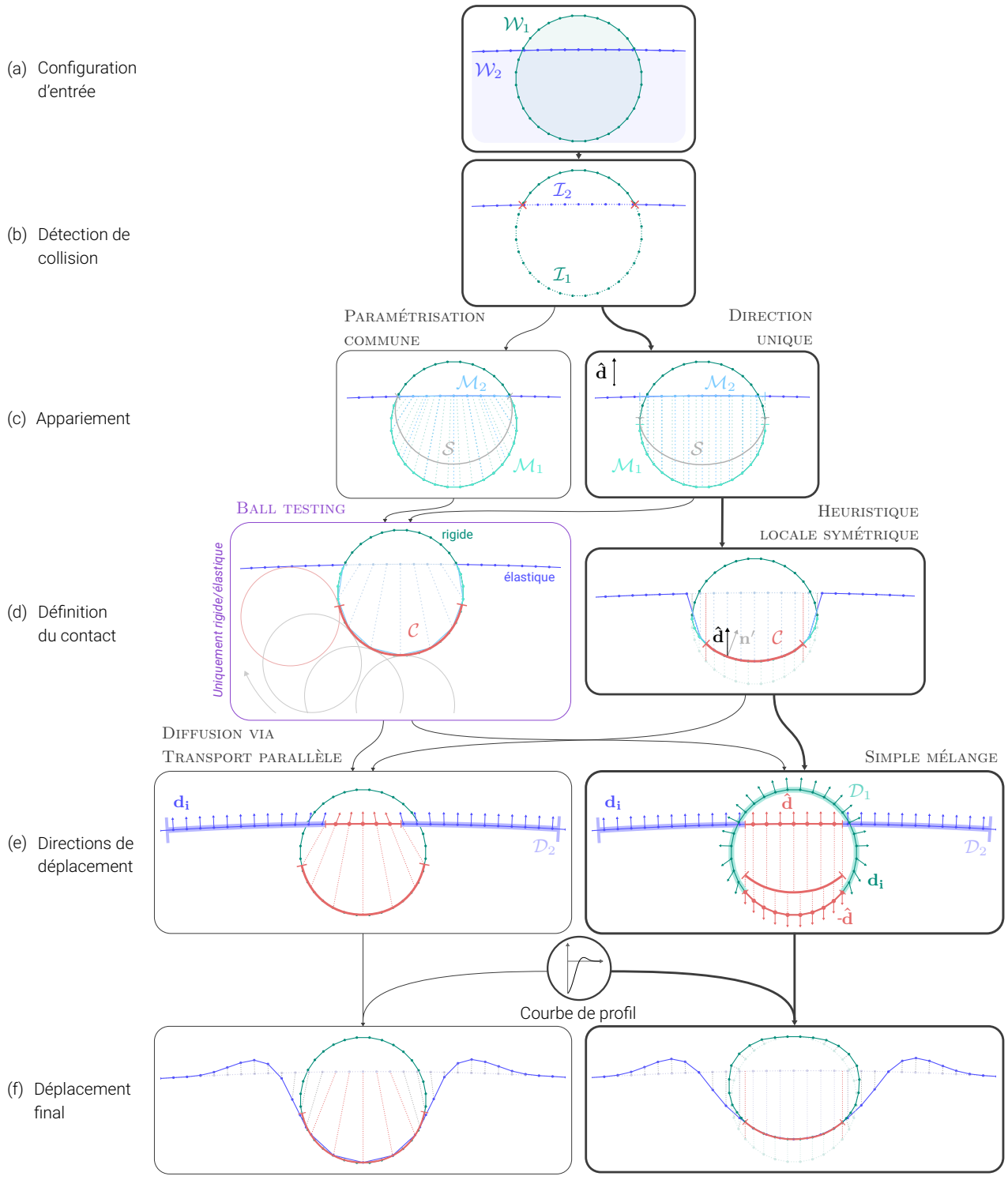


Figure D.2.: Aperçu de notre approche. Le pipeline développé, appliqué à chaque image sur chaque paire d'objets élastiques, peut être divisé en six étapes décrites ici de haut en bas. Différentes alternatives sont proposées pour certaines étapes mais le pipeline privilégié est mis en évidence en gras.

Appariement. Une sous-partie de ces zones d'intersection vont rester en contact entre elles à la fin de la déformation, il faut donc ensuite définir un appariement de ces zones pour définir la forme et l'étendue du contact pour chaque objet. En théorie, les définitions de l'appariement et du contact sont entremêlées, mais pour simplifier le problème, nous définissons d'abord un appariement conservatif entre les surfaces élastiques d'entrée, puis nous calculons les zones de contact restreintes.

La projection de ces régions sur le maillage opposé ne se trouve pas nécessairement à l'intérieur des régions en intersection et l'appariement peut donc s'étendre au-delà de l'intersection. Le sous-ensemble de chaque maillage l trouvant une correspondance sur la surface opposée fait partie des régions dites d'appariement $\mathcal{M}_l$ représentées en couleurs plus claires dans la Figure D.2(c). Nous avons développé deux stratégies différentes pour calculer cet appariement conservatif. La première, illustrée à gauche dans la Figure D.2(c), consiste à faire correspondre les régions d'intersection des deux maillages en espace paramétrique. Les directions d'appariement résultantes peuvent donc potentiellement varier pour chaque sommet des régions considérées. Dans cette version, les régions d'appariement $\mathcal{M}_l$ sont limitées par construction à $\mathcal{I}_l$. Cette méthode souffre donc de certaines limitations, telles que de fortes distorsions et étirements qui rendent difficile la définition du contact.

Pour pallier ces limitations, nous avons développé une seconde stratégie qui fait correspondre les deux surfaces au-delà des régions d'intersection. Nous calculons ces correspondances en trouvant pour chaque sommet de chaque maillage dans les régions d'intersection le point le plus éloigné sur la surface opposée le long d'une direction unique $\hat{\mathbf{d}}$. Pour éviter de considérer les maillages entiers, nous avons développé un algorithme efficace basé sur la silhouette de la surface vue depuis $\hat{\mathbf{d}}$ afin de restreindre de manière conservatrice le calcul de la correspondance.

Les deux méthodes produisent un champ de direction qui nous permet de projeter chaque surface sur la surface opposée pour résoudre la collision.

Surface potentielle de contact. À partir de cet appariement, nous déterminons ensuite la surface de contact potentielle $\mathcal{S}$, représentée en gris dans la Figure D.2(c). Elle est située par définition entre les deux surfaces appariées et sa forme et sa position dépendent du rapport des paramètres de pseudo-rigidité contrôlés par l'utilisateur pour chaque objet.

Comme le suggère son nom, $\mathcal{S}$ n'est pas la surface de contact finale mais correspond à la zone de contact maximale si les deux objets étaient suffisamment élastiques pour épouser entièrement la forme de l'objet opposé. Dans l'étape suivante, nous définirons la sous-partie de $\mathcal{S}$ qui restera effectivement en contact à la fin de la déformation.

Délimitation du contact. Après avoir projeté les sommets appariés des deux maillages sur $\mathcal{S}$, nous identifions la zone de contact $\mathcal{C}$, en rouge sur la Figure D.2(d). Notre approche générale est motivée par l'observation empirique que plus les objets sont rigides, plus les zones de contact devraient être petites.

Pour garantir la cohérence de l'animation malgré la contrainte d'indépendance tem-

porelle, $\mathcal{C}$ doit être définie de telle sorte à évoluer de manière continue sur chaque maillage pendant l'animation. En particulier, pour éviter les mouvements saccadés et les oscillations, nous ne pouvons pas les définir sur des éléments discrets tels que les sommets du maillage. Puisque la zone de contact est partagée par les deux surfaces en collision sur $\mathcal{S}$, la projection des zones délimitées sur chaque surface doit aboutir à la même zone de contact $\mathcal{C}$ sur $\mathcal{S}$.

Une fois encore, nous avons développé deux méthodes différentes pour cette étape. Nous proposons d'abord un algorithme, illustré dans la boîte violette de la Figure D.2(d), qui prend implicitement en compte la géométrie de $\mathcal{S}$, la profondeur de collision et un paramètre de pseudo-rigidité contrôlable par l'utilisateur.

Inspiré par les α-shapes [EM94], nous traduisons la rigidité d'un objet élastique par le rayon d'une balle roulant à l'intérieur de la surface projetée sur $\mathcal{S}$. La région considérée en contact est celle accessible par la balle, en rouge dans la Figure D.2(d). On observe que plus la balle est grosse, plus la région finale sera petite, et inversement, plus la balle est petite, plus la région finale de contact sera grande. Le rayon de la balle reflète donc la rigidité des objets. La limite de cette première méthode est sa définition asymétrique, ce qui la rend seulement pertinente pour le cas de la collision entre un objet rigide et un objet élastique.

Par conséquent, nous avons développé une seconde méthode basée sur la direction unique d'appariement $\hat{\mathbf{d}}$. Elle utilise un nouveau critère symétrique correspondant à nos observations et attentes concernant la définition de la zone de contact. Ce critère prend directement en compte la géométrie de $\mathcal{S}$ à travers son champ normal $\mathbf{n}'$, la profondeur de collision et un paramètre contrôlant l'étendue de la zone de contact. Cette méthode est illustrée dans la case de droite de la Figure D.2(d).

À la fin de cette étape, nous avons déterminé des régions continues et symétriques sur chaque maillage. Une fois projetées sur $\mathcal{S}$, elles définissent l'interface entre les deux objets à la fin de la déformation.

2.2. Déformation résultante

Après avoir défini la surface de contact, dans la deuxième étape de notre travail, nous avons pour objectif de déformer le voisinage $\mathcal{D}_l$ (en couleurs plus claires dans la Figure D.2(e)) de la zone de contact d'une manière plausible, indépendamment pour chaque objet l. Dans notre travail, la plausibilité sera obtenue en satisfaisant les trois contraintes suivantes :

- *Continuité de la surface déformée* : la déformation de la surface doit préserver la propriété de continuité du maillage initial.
- *Continuité pendant l'animation*: l'animation finale doit être fluide, sans mouvements saccadés ni oscillations pour donner l'illusion de fluidité et de continuité temporelle même si chaque image est traitée indépendamment.
- *Préservation du volume* : la déformation doit compenser le volume perdu par une surface fermée lors de la collision pour donner l'illusion de réalisme physique.

Nous considérons par défaut les déformations locales de surfaces fermées nécessitant une préservation du volume, mais si l'objet est un maillage ouvert, l'utilisateur doit avoir la possibilité d'ignorer partiellement ou entièrement cette contrainte afin d'obtenir des déformations plausibles. En plus de toutes ces exigences, nous pouvons rappeler l'un de nos principaux objectifs, à savoir le contrôle artistique.

Afin de simplifier le problème et d'obtenir un meilleur contrôle de la déformation, notre idée clé est de séparer la définition du déplacement final en deux parties : sa direction et sa magnitude. Nous définissons le déplacement de la surface initiale sur la région déformable $\mathcal{D}$ le long d'un champ de direction unitaire continu $\mathbf{d}$. Son amplitude est contrôlée par une courbe de profil 1D $\mathcal{H}$ instanciée à chaque sommet $\mathbf{p_i}$ de $\mathcal{D}$ et évaluée à l'aide d'une paramétrisation radiale unidimensionnelle u conduisant à sa position finale :

$$\mathbf{p}'_i = \mathbf{p}_i + \mathcal{H}_{a_i,s_i}(u_i)\,\mathbf{d}_i.$$

La forme de la courbe de profil $\mathcal{H}$ est paramétrée par l'amplitude a_i et la pente s_i à $u = 0$. Des degrés de liberté supplémentaires permettent un contrôle artistique étendu de cette courbe.

Région déformable. La région déformable $\mathcal{D}$ est définie comme le sous-ensemble de la région de travail qui sera déformé pour préserver le volume de l'objet initial. Comme indiqué précédemment, l'étendue de cette région dépend de la rigidité de cet objet. De plus, nous pouvons observer qu'elle doit évoluer avec la variation de la bordure de la région de contact $\partial\mathcal{C}$.

Pour à la fois délimiter et paramétrer cette région, nous calculons une paramétrisation radiale u à partir de $\partial\mathcal{C}$ sur l'ensemble de la région de travail, puis nous déterminons $\mathcal{D}$ en suivant le gradient de ce champ scalaire à partir de $\partial\mathcal{C}$. Intuitivement, u localise tout point de la surface le long d'une géodésique continue allant de $\partial\mathcal{C}$ à la limite externe de la région déformable $\mathcal{D}$. L'étendue de la région déformable est déterminée par une distance pseudo-géodésique contrôlée par l'utilisateur.

Il est intéressant de noter que, comme $\mathcal{C}$ change à chaque image, la paramétrisation u et, par construction, la région $\mathcal{D}$ doivent être recalculées dynamiquement à chaque image. Notons également que, puisque leur définition est basée sur $\partial\mathcal{C}$ qui est temporellement continue, elles évoluent aussi de manière continue au cours de l'animation.

Champ de direction. Le champ de direction unitaire $\mathbf{d}$ le long duquel la surface sera déplacée est soumis à deux contraintes : (1) il doit correspondre aux déplacements fixes de la zone de contact sur la surface de contact potentielle $\mathcal{S}$ et, (2) pour garantir la contrainte de continuité géométrique, il doit être principalement aligné avec les normales initiales de la surface. Nous avons fait l'observation que, lors d'une collision, un objet élastique est déformé en dehors de la zone d'impact le long de son champ de normales, formant un gonflement. Par conséquent, l'idée est de faire converger rapidement les directions vers les normales initiales à mesure que l'on s'éloigne de $\partial\mathcal{C}$.

L'approche la plus naturelle, mais coûteuse, pour résoudre ce problème consiste à diffuser par transport parallèle la déviation des directions d'appariement contraintes par rapport aux normales de la surface initiale le long de $\partial\mathcal{C}$ avec la contrainte de s'annuler

à la limite extérieure de la région déformable. Le champ de direction unitaire résultant est illustré par les flèches à gauche de la Figure D.2(e).

Une façon plus simple de résoudre le même problème est de diffuser la direction d'appariement contrainte sur toute la région déformable $\mathcal{D}$ et de la mélanger avec les normales de la surface initiale en utilisant une fonction de transfert non linéaire se basant sur champ scalaire u. Cette méthode est illustrée dans la case de droite de la Figure D.2(e).

Définition de la courbe de profil. Concernant la magnitude du déplacement, nous définissons une famille de fonctions, appelées *courbes de profil*, instanciées en chaque sommet de la région déformable. Les degrés de liberté correspondant à l'amplitude a_i et à la pente s_i à $\partial\mathcal{C}$ (i.e., $u = 0$) sont nécessaires pour assurer respectivement la continuité C^0 et C^1 avec la zone de contact. Pour pouvoir évaluer $\mathcal{H}$ partout, a et s sont diffusées sur toute la région déformable $\mathcal{D}$. Trois autres degrés de liberté sont nécessaires pour garantir la continuité C^2 avec la partie non déformée du maillage initial et pour s'assurer que la déformation diminue de manière continue à la limite extérieure de la région déformable. Enfin, la courbe de profil doit posséder un dernier degré de liberté pour contrôler linéairement le gonflement et assurer la préservation exacte du volume.

Le principal avantage de cette définition est qu'elle ouvre la porte à un large éventail de contrôles artistiques sur la réponse à la collision. Le gonflement peut être contrôlé artistiquement pour exagérer ou annuler la préservation de volume. Il peut même être déplacé le long de la paramétrisation u ou étalé pour modifier la répartition du volume. Il peut également être réparti de manière anisotrope sur $\mathcal{D}$ pour produire une réponse de déformation plus plausible. On peut même imaginer produire des rides en ajoutant à la courbe de profil des détails à haute fréquence, comme le montre la Figure D.3.

Les figures D.4 à D.6 présentent des résultats obtenus avec notre méthode. Notre pipeline peut être appliqué sur des objets constitués de plusieurs composantes connexes distinctes (Figure D.4) ou même des configurations avec des surfaces bosselées (Figure D.5). En Figure D.6, nous comparons notre méthode à une simulation par éléments finis (FEM) réalisée avec Houdini© SideFX et appliquée à deux balles élastiques en collision. Notre déformeur est capable de reproduire fidèlement les déformations et les effets de gonflement observés dans la simulation.

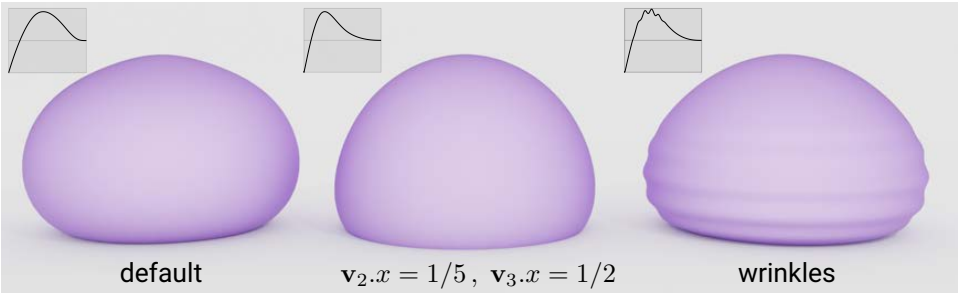


Figure D.3.: Variations artistique du profil. Nous visualisons ici l'effet de la modification des abscisses de $\mathbf{v}_2$ et $\mathbf{v}_3$ (au milieu), ou de l'ajout d'une fonction sinusoïdale pour imiter les rides (à droite). Les courbes de profil respectives pour une pente moyenne sont indiquées dans les coins.

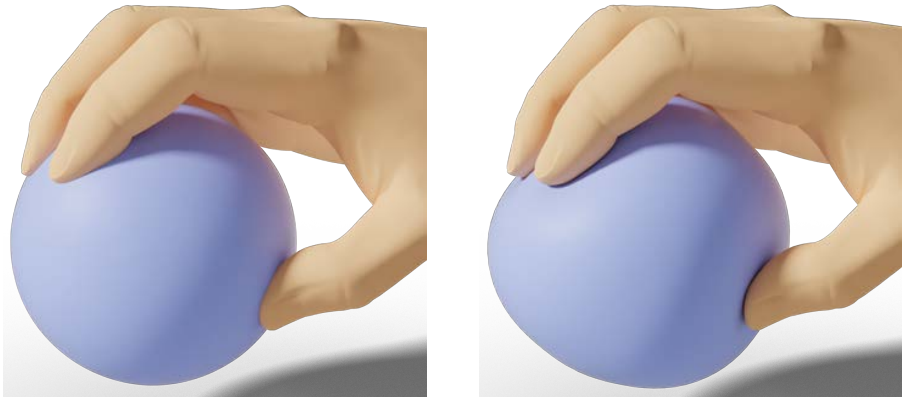


Figure D.4.: Composantes multiples. À gauche : configuration initiale, chaque doigt correspond à une région de travail indépendante. À droite: deformation finale. Nous pouvons remarquer le gonflement subtil du bout des doigts, par exemple du pouce sur la balle.

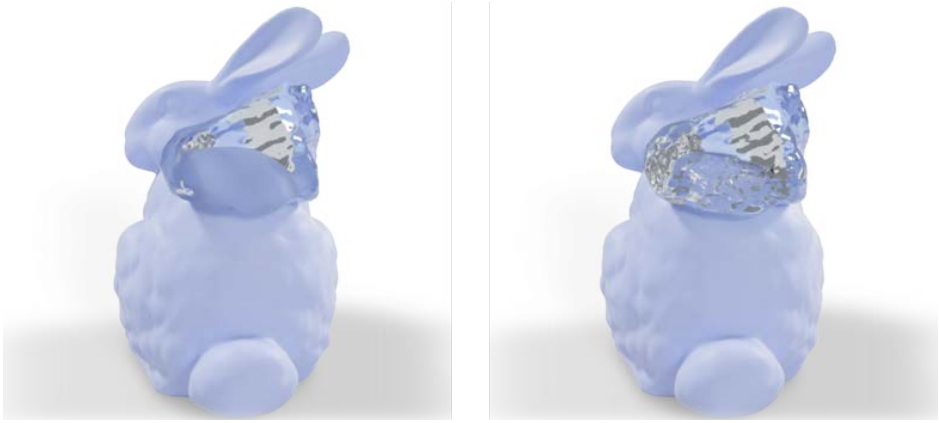


Figure D.5.: Surfaces bosselées. À gauche : configuration initiale. À droite : deformation finale. L'objet gélifié est coupé pour montrer la surface de contact.

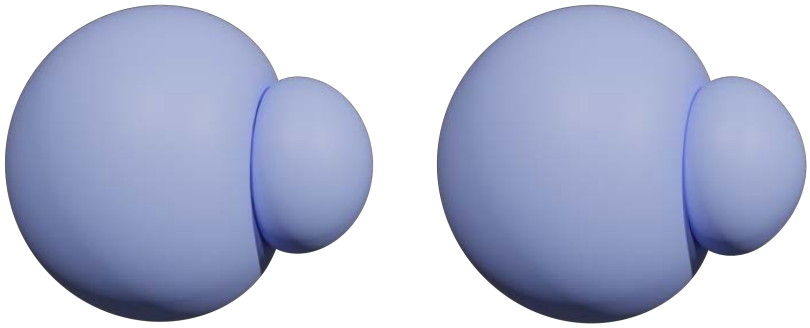


Figure D.6.: Comparaison avec une simulation. À gauche: simulation FEM dans Houdini©SideFX. À droite: notre méthode.

2.3. Application au skinning

Le pipeline décrit jusqu'ici suppose que les régions de travail impliquées dans la collision sont distinctes, sans chevauchement, et que les déformations sont nulles à leurs bordures externes. Cependant, de telles hypothèses ne sont pas toujours valides. C'est typiquement le cas d'une articulation pour laquelle la peau associée aux deux parties adjacentes entre en collision (par exemple, le coude, le genou, etc.). Dans de telles configurations, la déformation ne devrait pas être nulle entre les deux zones considérées et il est même impossible de définir une frontière claire séparant ces deux parties.

Nous adressons cette limitation en permettant aux régions de travail adjacentes de se chevaucher, tout en assurant la continuité spatiale par un mélange de leur déformation respective sur ce qu'on appelle la *région partagée* $\mathcal{R}$. Soit β les poids de mélange formant une partition de l'unité, alors la position finale $\mathbf{p}'_i$ de chaque point dans $\mathcal{R}$ est obtenue par:

$$\mathbf{p}'_i = \beta_i \, \mathbf{p}'_{i,1} + (1 - \beta_i) \, \mathbf{p}'_{i,2} \ ,$$

où $\mathbf{p}'_{i,l}$ correspond à la position après déplacement obtenue en appliquant notre pipeline à la région de travail $\mathcal{W}_l$.

Cette approche est très simple mais nécessite quelques traitements particuliers supplémentaires. Nous devons notamment calculer automatiquement un partitionnement strict des régions de travail afin que le calcul des surfaces de contact (Section 2.1) puisse être effectué. Le découpage virtuel doit suivre le pli de l'articulation qui varie dans le temps. Par exemple, au cours d'une animation, le pli d'un coude ne correspond pas toujours à la même bande de triangles et se déplace légèrement sur la surface initiale. Il est donc impossible pour un artiste de fixer en pré-traitement un emplacement pour ce pli qui soit valable pour tout mouvement de l'articulation. Nous avons donc développé une méthode automatique, basée sur la détection de contours et le champ scalaire β, pour calculer le partitionnement des régions de travail à la volée à chaque image.

De plus, nous avons introduit des points de synchronisation pendant le calcul des paramètres de déformation de chaque région de travail afin que les positions intermédiaires $\mathbf{p}'_{i,l}$ soient déjà aussi proches que possible les unes des autres avant d'en faire la moyenne.

La Figure D.7 montre un résultat obtenu avec cette extension de notre pipeline.

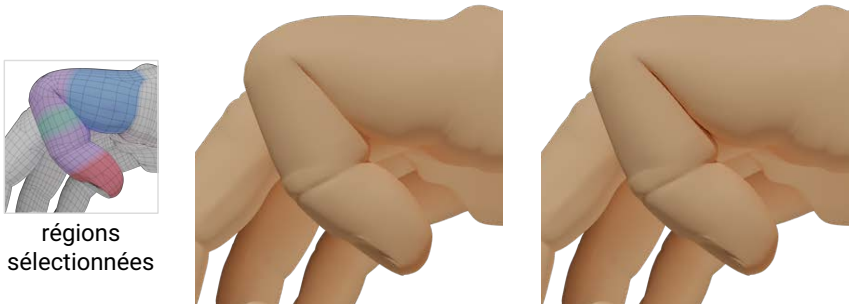


Figure D.7.: Application au skinning. À gauche: configuration initiale et zones de travail correspondantes. Les régions partagées (en violet) se trouvent entre les phalanges. Chaque phalanges correspond à une zone de tavail différentes. À droite: déformation finale.

3. Conclusion et perspectives

Tout au long de cette thèse, nous avons cherché à développer une technique de déformation pour les objets élastiques en collision qui fournit des retours interactifs à l'utilisateur avec des paramètres qui peuvent être contrôlés intuitivement. Nous ne recherchions pas le réalisme physique, mais plutôt des résultats plausibles, produits rapidement, intuitivement et avec des retours en temps réel pour le domaine de l'animation par ordinateur.

Dans ce manuscrit, nous avons présenté une nouvelle technique de déformation pour la résolution des contacts élastiques-élastiques locaux tout en produisant des effets de gonflement plausibles et en offrant un contrôle artistique étendu. Cette technique ne vise pas à générer de grandes déformations globales qui devraient déjà être traitées par le *skinning*, les *Blend Shapes* ou les déformeurs basées sur les cages. En nous appuyant entièrement sur des opérations géométriques, nous avons réussi à obtenir un retour instantané, sans dépendance temporelle, pour une intégration transparente dans le pipeline d'animation. La continuité temporelle au cours d'une animation est assurée sans introduire de dépendances temporelles, grâce à la définition de zones continues à chaque étapes. Enfin, en combinant une paramétrisation 1D et une courbe de profil 2D, nous sommes parvenus à produire des gonflements plausibles préservant le volume initial des objets considérés. Cette association permet également un contrôle artistique du gonflement, allant de la préservation totale du volume à son exagération, ainsi qu'une large gamme de détails qui peuvent être introduits à l'aide d'outils artistiques.

Nous avons également proposé une extension de notre pipeline pour gérer les auto-intersections locales, ce qui permet d'offrir la gestion des contacts aux méthodes de skinning classiques avec un contrôle du gonflement et autres effets artistiques.

Même si nous ne nous sommes pas appuyés sur les lois de la physique comme les méthodes classiques de simulation physique, la plausibilité de nos déformations finales est renforcée par une répartition anisotrope du volume réinjecté. Par ailleurs, en plus de pouvoir transmettre des informations sur les propriétés physiques des objets en utilisant peu de paramètres, notre approche permet à l'artiste de concevoir des effets cartoonesques en utilisant l'exagération de la compensation de volume, mais également en jouant sur sa répartition en utilisant l'un des nombreux contrôles artistiques que nous avons proposés : manipulation globale ou locale de la courbe de profil, paramètres manuels constant ou variant dans l'espace, etc.

Ce type d'approche ouvre de nouvelles perspectives pour la déformation d'objets élastiques en contact dans le contexte de l'animation par ordinateur : l'édition peut être effectuée de manière non-linéaire, des déformations similaires peuvent être reproduites dans différents contextes sans nécessiter de fastidieux ajustements manuels, et de nombreux effets artistiques peuvent être facilement réalisés en manipulant la courbe du profil.

Au delà des améliorations propres à notre méthode, nous avons imaginés plusieurs perspectives plus générales.

Étude utilisateur. Tout d'abord, comme nous avons développé un outil pour les artistes, il serait intéressant d'avoir leur avis sur l'utilisation et l'utilité de ce travail par le biais d'une étude utilisateur. Cela implique toutefois que l'on ait accès à des utilisateurs

amateurs et professionnels de tels outils. Dans le même ordre d'idée, il serait intéressant de tester notre méthode sur de véritables rigs de production. Presque tous les résultats de cette thèse ont été riggés et animés par des artistes non-professionnels. De plus, des comparaisons avec des simulations ou des déformations capturées, plus complexes et conçues par des professionnels pourraient également être utiles pour valider la plausibilité des déformations produites par notre méthode.

Élasticité de la peau. Pour améliorer la plausibilité de nos déformations, même dans des configurations cartoonesques extrêmes, nous pourrions également prendre en compte davantage de propriétés physiques de la peau, telle que son élasticité en plus de sa rigidité déjà considérée. En effet, les déplacements sont actuellement effectués pour chaque sommet le long d'un champ de direction sans tenir compte de l'étirement que peut subir la peau lors de l'application de la déformation. Cet étirement peut être encore aggravé par l'exagération artistique de la préservation du volume. Il faudrait donc permettre des mouvements tangentiels des sommets pour limiter ces étirements et trouver un moyen direct de les intégrer dans notre méthode.

Détails procéduraux. Lorsque la peau est déformée, la dilatation ou la contraction de la méso-géométrie de la surface et de la microstructure du matériau peut modifier de manière significative les propriétés de réflexion de la surface une fois intégrées dans les modèles d'éclairage. Par exemple, les plis de la peau, y compris les rides, peuvent apparaître ou disparaître pendant la déformation. Certaines parties peuvent également passer du rouge au blanc et vice-versa en raison des variations de la concentration de sang dans les tissus. Une déformation à faible échelle issue du skinning pourrait donc être complétée par de tels détails et effets haute-fréquence. Une combinaison d'approches procédurales avec un modèle d'apparence multi-résolution pourrait être intéressante à étudier. Une autre solution plus simple pour ajouter des détails avec notre implémentation actuelle pourrait être de peindre directement sur la surface des détails cachés qui seraient révélés lors du processus de collision, par exemple des rides devant un doigt glissant sur un avant-bras ou au début d'un pincement de la peau.

Nouvelle méthode de skinning. Comme il est appliqué en tant que post-traitement, le résultat de notre pipeline dépend fortement de la méthode de skinning géométrique utilisée et des déformations de maillage qui en résultent. Les techniques actuelles de skinning et les paramètres de rigging qui leur sont associés sont conçus pour empêcher les auto-intersections en rendant les déformations anormalement souples et donc peu réalistes. Pour exploiter pleinement le potentiel de notre déformeur, il serait pertinent d'étudier de nouvelles techniques de skinning et de rigging axées sur la génération de belles déformations sur la partie extérieure visible de l'articulation, tout en produisant délibérément des auto-intersections à l'intérieur de l'articulation que pourra résoudre notre déformeur. De plus, notre déformation préservant le volume ne corrige pas le volume initial perdu par la méthode de skinning et, même si nous sommes capables de calculer cette perte, notre déformeur n'est pas conçu pour la corriger. Afin d'avoir une préservation complète

du volume à partir de la pose de repos d'un modèle articulé, la solution serait soit de concevoir une méthode de skinning préservant automatiquement le volume, soit avant l'application de notre déformeur, d'appliquer une correction de volume en utilisant une méthode comme celles de von Funck et al. [vFTS08] ou Rohmer et al. [RHC09].

Autres principes d'animation. Enfin, durant cette thèse, nous n'avons considéré que (la moitié de) l'un des douze principes d'animation présentés par Disney : *le squash* (de "Squash and stretch"). Un défi passionnant serait d'étudier d'autres principes en appliquant la même philosophie qui consiste à fournir à l'artiste des outils automatiques intuitifs mais à vocation artistique pour produire les effets associés. Récemment, Zang et al. [ZBLJ20] et Rohmer et al. [RTK+21] ont proposé deux méthodes pour ajouter des effets de mouvements secondaires sur un personnage animé. La première méthode utilise des simulations produisant des mouvements orthogonaux à la déformation issue du skinning, tandis que la seconde propose une solution géométrique s'appuyant sur le mouvement du squelette. Les deux approches produisent des animations moins robotiques et plus vivantes sans nécessiter une simulation complète lourde et difficilement dirigeable par l'artiste, appliquée après l'étape d'animation et de rigging.

Toutes ces approches ouvrent la voie à de nouvelles façons de créer les animations de demain.

Bibliography

[ACWK06] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. Swirling-sweepers: Constant-volume modeling. *Graph. Models*, 68(4):324–332, 2006.

[AFC+10] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G. Kry. Volume contact constraints at arbitrary resolution. *ACM Trans. Graph.*, 29(4), July 2010.

[AOW+08] Bart Adams, Maks Ovsjanikov, Michael Wand, Hans-Peter Seidel, and Leonidas J. Guibas. Meshless Modeling of Deformable Shapes and their Motion. In Markus Gross and Doug James, editors, *Eurographics/SIGGRAPH Symposium on Computer Animation*. The Eurographics Association, 2008.

[APHS11] Garrett Aldrich, Dmitriy V Pinskiy, Bernd Hamann, and Walt Disney Animation Studios. Collision-driven volumetric deformation on the gpu. In *Eurographics (Short Papers)*, pages 9–12, 2011.

[ARF15] Nadine Abu Rumman and Marco Fratarcangeli. Position-based skinning for soft articulated characters. *Computer Graphics Forum*, 34(6):240–250, 2015.

[AS07] Alexis Angelidis and Karan Singh. Kinodynamic skinning using volume-preserving deformations. In *Proc. of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 129–140. Eurographics Association, 2007.

[ATW+17] Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi. Sketch-based implicit blending. *ACM Trans. Graph.*, 36(6), 2017.

[AWC06] Alexis Angelidis, Geoff Wyvill, and Marie-Paule Cani. Sweepers: Swept deformation defined by gesture. *Graph. Models*, 68(1):2–14, January 2006.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[Ber97] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of graphics tools*, 2(4):1–13, 1997.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, July 2002.

[BJ10] Jernej Barbič and Doug L. James. Subspace self-collision culling. *ACM Trans. on Graphics (SIGGRAPH 2010)*, 29(4):81:1–81:9, 2010.

[BK01] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *Proceedings of the Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 359–374. Springer-Verlag, 2001.

[BK04] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, 2004.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, page 28–36, Goslar, DEU, 2003. Eurographics Association.

[BML+14] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11, 2014.

[BMM15] Jan Bender, Matthias Müller, and Miles Macklin. Position-based simulation methods in computer graphics. In *EG 2015 - Tutorials*. The Eurographics Association, 2015.

[BS07] Alexander I. Bobenko and Boris A. Springborn. A discrete laplace–beltrami operator for simplicial surfaces. *Discrete Comput. Geom.*, 38(4):740–756, December 2007.

[BS08] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, January 2008.

[BT95] Srikanth Bandi and Daniel Thalmann. An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies. *Computer Graphics Forum*, 1995.

[CB17] Stéphane Calderon and Tamy Boubekeur. Bounding proxies for shape approximation. *ACM Trans. Graph.*, 36(4), 2017.

[CWW13] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.*, 32(5):152:1–152:11, 2013.

[DB13] Crispin Deul and Jan Bender. Physically-based character skinning. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Eurographics Association, 2013.

[DdL13] Olivier Dionne and Martin de Lasa. Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, page 173–180, New York, NY, USA, 2013. Association for Computing Machinery.

[EM94] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, 1994.

[Eri04] Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc., 2004.

[FKN80] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '80, page 124–133, New York, NY, USA, July 1980. Association for Computing Machinery.

[Flo03] Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[GLM96] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996.

[GMS14] Ming Gao, Nathan Mitchell, and Eftychios Sifakis. Steklov-poincaré skinning. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 139–148. Eurographics Association, 2014.

[Gra06] L. Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.

[HA18] Philipp Herholz and Marc Alexa. Factor once: Reusing cholesky factorizations on sub-meshes. *ACM Transaction on Graphics*, 37(6), 2018.

[HA19] Philipp Herholz and Marc Alexa. Efficient computation of smoothed exponential maps. *Computer Graphics Forum*, 38:79–90, 2019.

[HBB+12] Stefanie Hahmann, Georges-Pierre Bonneau, Sébastien Barbier, Gershon Elber, and Hans Hagen. Volume Preserving FFD for Programmable Graphics Hardware. *Visual Computer*, 28(3):231–245, February 2012.

[HDA17] Philipp Herholz, Timothy A. Davis, and Marc Alexa. Localized solutions of sparse linear systems for geometry processing. *ACM Transactions on Graphics*, 36(6), 2017.

[HF07] Marco Hutter and Arnulph Fuhrmann. Optimized continuous collision detection for deformable triangle meshes. *J. WSCG*, 15:25–32, July 2007.

[HPSZ11] David Harmon, Daniele Panozzo, Olga Sorkine, and Denis Zorin. Interference-aware geometric modeling. *ACM Trans. Graph.*, 30(6):137:1–137:10, 2011.

[HTG03] Bruno Heidelberger, Matthias Teschner, and Markus Gross. Real-time volumetric intersections of deforming objects. In *VMV*, volume 2003, pages 461–468, 01 2003.

[HTG04] Bruno Heidelberger, Matthias Teschner, and Markus Gross. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG*, 12(3):145–152, 01 2004.

[Hub95] Philip Martyn Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[HZ00] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, page 517–526, 2000.

[JBPS11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4), July 2011.

[JMD+07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3):71–es, 2007.

[JP04] Doug L. James and Dinesh K. Pai. Bd-tree: Output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph.*, 23(3):393–398, August 2004.

[KB18] Martin Komaritzan and Mario Botsch. Projective skinning. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(1):12:1–12:19, 2018.

[KB19] Martin Komaritzan and Mario Botsch. Fast projective skinning. In *Motion, Interaction and Games*, MIG '19. ACM, 2019.

[KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4):59:1–59:10, 2013.

[KCS13] Mathieu Desbrun Keenan Crane, Fernando de Goes and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM.

[KCvO08] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, 27(4):105:1–105:23, 2008.

[KHM+98] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, January 1998.

[KS12] Ladislav Kavan and Olga Sorkine. Elasticity-inspired deformers for character articulation. *ACM Trans. Graph.*, 31(6):196:1–196:8, 2012.

[KZ05] Ladislav Kavan and Jiri Zara. Spherical blend skinning: A real-time deformation of articulated models. In *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 9–16. ACM Press, April 2005.

[LAM03] Thomas Larsson and Tomas Akenine-Möller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19:164–174, 05 2003.

[Las87] John Lasseter. Principles of traditional animation applied to 3d computer animation. *SIGGRAPH Comput. Graph.*, 21(4):35–44, 1987.

[LB19] Yijing Li and Jernej Barbič. Multi-resolution modeling of shapes in contact. *Proc. ACM Comput. Graph. and Interact. Tech.*, 2(2), 2019.

[LC91] Ming C Lin and John F Canny. A fast algorithm for incremental distance calculation. In *ICRA*, volume 91, pages 9–12. Citeseer, 1991.

[LCF00] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 165–172. ACM, 2000.

[LFS+20] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Transactions on Graphics*, 39(4), 2020.

[LGS+09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast bvh construction on gpus. *Computer Graphics Forum*, 2009.

[LH16] Binh Huy Le and Jessica K. Hodgins. Real-time skeletal skinning with optimized centers of rotation. *ACM Trans. Graph.*, 35(4):37:1–37:10, 2016.

[LL19] Binh Huy Le and J P Lewis. Direct delta mush skinning and variants. *ACM Trans. Graph.*, 38(4):113:1–113:13, 2019.

[LVGO21] Binh Huy Le, Keven Villeneuve, and Carlos Gonzalez-Ochoa. Direct delta mush skinning compression with continuous examples. *ACM Trans. Graph.*, 40(4), 2021.

[MCC11] Tim McLaughlin, Larry Cutler, and David Coleman. Character rigging, deformations, and simulations in film and game production. In *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery.

[MCKM15] Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. Air meshes for robust collision handling. *ACM Trans. Graph.*, 34(4), July 2015.

[MDRW14] Joe Mancewicz, Matt L. Derksen, Hans Rijpkema, and Cyrus A. Wilson. Delta mush: Smoothing deformations while preserving detail. In *Proceedings of the Fourth Symposium on Digital Production*, DigiPro '14, page 7–11, New York, NY, USA, 2014. Association for Computing Machinery.

[Mir97] Brian Mirtich. Efficient algorithms for two-phase collision detection. Technical Report TR-97-23, Mitsubishi Electric Research Laboratory, 1997.

[Mir98] Brian Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions On Graphics (TOG)*, 17(3):177–208, 1998.

[MOK95] Karol Myszkowski, Oleg G Okunev, and Tosiyasu L Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–511, 1995.

[MSJT08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: Class notes. In *ACM SIGGRAPH 2008 Classes*. ACM, 2008.

[MTLT88] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88*, pages 26–33. Canadian Information Processing Society, 1988.

[MZS+11] Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30(4):37:1–37:12, 2011.

[NMK+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[NS13] Jesús R. Nieto and Antonio Susín. *Cage Based Deformations: A Survey*, pages 75–99. Springer, 2013.

[PCYQ18] Junjun Pan, Lijuan Chen, Yuhan Yang, and Hong Qin. Automatic skinning and weight retargeting of articulated characters using extended position-based dynamics. *Vis. Comput.*, 34(10):1285–1297, October 2018.

[PPG04] Mark Pauly, Dinesh K. Pai, and Leonidas J. Guibas. Quasi-rigid objects in contact. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 109–119. Eurographics Association, 2004.

[RARC+18] Valentin Roussellet, Nadine Abu Rumman, Florian Canezin, Nicolas Mellado, Ladislav Kavan, and Loic Barthe. Dynamic implicit muscles for character skinning. *Computers and Graphics*, 77:227–239, 2018.

[RHC09] Damien Rohmer, Stefanie Hahmann, and Marie-Paule Cani. Exact volume preserving skinning with shape control. In *Proc. of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 83–92. ACM, 2009.

[Riv84] Maria-Cecilia Rivara. Mesh Refinement Processes Based on the Generalized Bisection of Simplices. *SIAM Journal on Numerical Analysis*, 21(3):604–613, jun 1984.

[RTK+21] Damien Rohmer, Marco Tarini, Niranjan Kalyanasundaram, Faezeh Moshfeghifar, Marie-Paule Cani, and Victor Zordan. Velocity Skinning for Real-time Stylized Skeletal Animation. *Computer Graphics Forum*, 2021.

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, page 109–116, Goslar, DEU, 2007. Eurographics Association.

[SB09] Olga Sorkine and Mario Botsch. Interactive Shape Modeling and Deformation. In K. Museth and D. Weiskopf, editors, *Eurographics 2009 - Tutorials*. The Eurographics Association, 2009.

[SdGK18] Breannan Smith, Fernando de Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Trans. Graph.*, 37(2):12:1–12:15, 2018.

[SF91] Mikio Shinya and Marie-Claire Forgue. Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, 2(4):132–134, 1991.

[SGHS98] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, 1998.

[SKSJ20] Silvia Sellán, Jacob Kesten, Ang Yan Sheng, and Alec Jacobson. Opening and closing surfaces. *ACM Transactions on Graphics*, 2020.

[SP86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.

[TCYM09] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. Iccd: Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):544–557, 2009.

[THM+03] Matthias Teschner, Bruno Heidelberger, Matthias Mueller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. In *VMV*, pages 47–54, 2003.

[TJ81] F. Thomas and Ollie Johnston. The illusion of life : Disney animation. 1981.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[TOK14] Yun Teng, Miguel A. Otaduy, and Theodore Kim. Simulating articulated subspace self-contact. *ACM Trans. Graph.*, 33(4):106:1–106:9, 2014.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH Comput. Graph.*, 21(4):205–214, 1987.

[Tur90] Greg Turk. Interactive collision detection for molecular graphics. Technical Report TR90-014, University of North Carolina at Chapel Hill, USA, 1990.

[VBG+13] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Trans. Graph.*, 32(4):125:1–125:12, 2013.

[vFTS06] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[vFTS08] Wolfram von Funck, Holger Theisel, and Helmut Seidel. Volume-preserving mesh skinning. In *Proc. of the Vision, Modeling, and Visualization Conference*, 2008.

[VGB+14] Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. Robust iso-surface tracking for interactive character skinning. *ACM Trans. Graph.*, 33(6):189:1–189:11, 2014.

[VJ94] George Vaněkčkek Jr. Back-face culling applied to collision detection of polyhedra. *The Journal of Visualization and Computer Animation*, 5(1):55–63, 1994.

[VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum*, 20(3):260–267, 2001.

[VT94] Pascal VOLINO and Nadia Magnenat THALMANN. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, 1994.

[Wan15] Wei Wang. A collision deformer for autodesk maya. Master's thesis, Texas A & M University, 2015.

[WC21] Monan Wang and Jiaqi Cao. A review of collision detection for deformable objects. *Computer Animation and Virtual Worlds*, page e1987, 2021.

[WDZ17] Rene Weller, Nicole Debowski, and Gabriel Zachmann. kdet: Parallel constant time collision detection for polygonal objects. *Computer Graphics Forum*, 36:131–141, 05 2017.

[WJBK15] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. *ACM Trans. Graph.*, 34(4), July 2015.

[WLT+17] Tongtong Wang, Zhihua Liu, Min Tang, Ruofeng Tong, and D. Manocha. Efficient and reliable self-collision culling using unprojected normal cones. *Computer Graphics Forum*, 36, 2017.

[WWB+14] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4), July 2014.

[YKH+10] Sung-eui Yoon, Young J. Kim, Takahiro Harada, Young J. Kim, and Sung-eui Yoon. Recent advances in real-time collision and proximity computations for games and simulations. In *ACM SIGGRAPH ASIA 2010 Courses*, SA '10, New York, NY, USA, 2010. Association for Computing Machinery.

[Zac98] Gabriel Zachmann. Rapid collision detection by dynamically aligned dop-trees. In *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No. 98CB36180)*, pages 90–97. IEEE, 1998.

[ZBLJ20] Jiayi Eris Zhang, Seungbae Bang, David I. W. Levin, and Alec Jacobson. Complementary dynamics. *ACM Trans. Graph.*, 39(6), 2020.

Déformations géométriques indépendantes du temps pour les contacts élastiques

Résumé : Les films et séries d'animation étant de plus en plus présents dans le divertissement grand public, les besoins des artistes en terme d'outils d'animation rapides et intuitifs ne cessent de croître. Les artistes ne font pas seulement appel à leur imagination et à leurs compétences pour donner vie à aux modèles numériques, ils s'inspirent également du monde physique pour mieux immerger les spectateurs dans leur environnement virtuel.

De nombreux objets de notre environnement quotidien présentent des déformations élastiques lorsqu'ils sont mis en contact avec d'autres, par exemple une balle anti-stress écrasée par une main, un oreiller écrasant une tête lors d'une bataille d'oreillers ou un ballon rebondissant sur un poteau de but. Ils ont notamment tendance à s'écraser à l'intérieur du contact et à gonfler à l'extérieur. Ces effets d'écrasement et de gonflement sont essentiels pour communiquer une déformation plausible tout en capturant le comportement physique des matériaux mous dans divers contextes, tels que les films d'animation. Ce type de déformation est toutefois connu pour être difficile et fastidieux à reproduire manuellement par les artistes, et les outils existants restent limités pour une utilisation artistique.

En pratique, ces déformations sont donc générées par des méthodes de simulation physique. Cependant, en raison de leur dépendance temporelle, elles doivent être exécutées après les étapes de rigging et d'animation, ce qui empêche une édition non linéaire de la scène 3D. De plus, les artistes ont souvent recours à des effets de déformation caricaturaux pour mieux transmettre les émotions et les idées qui sont difficiles à obtenir par simulation physique.

La principale contribution de cette thèse est un nouvel outil de déformation purement géométrique et indépendant du temps qui assiste l'artiste en résolvant les contacts locaux entre les objets élastiques, ainsi qu'en produisant des effets de gonflement qui peuvent être contrôlés par l'artiste. Pour parvenir à une intégration transparente dans le processus de création d'animation, nous avons conçu notre outil de déformation de manière à fournir un retour instantané à l'artiste tout en permettant une édition non linéaire grâce à une stratégie entièrement indépendante du temps. Pour produire des effets de gonflement plausibles, notre méthode peut aussi préserver intégralement le volume, bien que des contrôles artistiques soient également possibles pour explorer des comportements plus exagérés. Plus précisément, à partir de plusieurs maillages en intersection, notre déformeur calcule d'abord les parties des surfaces restant en contact, puis applique un déplacement procédural contrôlé par une courbe de profil. Même si notre outil traite chaque image indépendamment, il réalise des déformations temporellement continues avec un contrôle artistique du gonflement grâce à un petit nombre de paramètres de pseudo-rigidité. La plausibilité de la déformation est encore renforcée par la répartition anisotrope du gonflement préservant le volume. Une extension est également proposée pour gérer les auto-collisions entre des parties adjacentes d'un même objet, qui se produisent fréquemment dans le contexte d'animation de personnages.

Le résultat de ce travail est un déformeur temps réel robuste qui permet de gérer des configurations géométriques complexes telles qu'une balle écrasée par une main, des lèvres qui se touchent, des doigts qui se plient, etc.

Mots-clés : Animation 3D, deformation de maillages, resolution de collision, informatique graphique, traitement de la géométrie.

Time-independent geometrical deformation for elastic contacts

Abstract: As animated films and series become more and more present in the mainstream entertainment, the artists' needs are growing in term of fast and intuitive animation tools. Artists not only heavily rely on their imagination and skills to bring digital models to life; they also take inspiration from the physical world to better immerse viewers in their virtual environment.

Many objects of our everyday surroundings exhibit elastic deformations when put in contact with others, e.g., a stress ball crushed by a hand, a pillow smashing a head during a pillow fight or a soft ball bouncing on a goal post. They most notably tend to squash inside the contact and to bulge outside of it. Such squashing and bulging effects are essential to communicate plausible deformation while capturing the physical behavior of soft materials in a variety of contexts, such as animated films. This type of deformation is, however, notoriously difficult and tedious to manually reproduce by computer graphics (CG) artists, and existing tools remain limited for artistic use.

In practice, such deformations are thus generated through physically based simulation methods. However, due to their time-dependency, physical simulations must be run after the rigging and animation steps, preventing non-linear editing of the 3D scene. Moreover, artists also often resort to cartoonish deformation effects to better convey emotions and thoughts. Such exaggerated effects are difficult to achieve through physical simulations.

The main contribution of this thesis is a novel purely geometric deformation framework that assists the artist by resolving local contacts between elastic objects and producing bulge effects in an art-directable way. To achieve a seamless integration within animation workflows, we designed our deformation tool to provide instant feedback to the artist while enabling non-linear editing thanks to a fully time-independent strategy. To produce plausible bulge effects, our method can also preserve the volume exactly, while artistic controls are also possible to explore more exaggerated behaviors. More specifically, starting from multiple meshes in intersection, our deformer first computes the parts of the surfaces remaining in contact, and then applies a procedural displacement controlled by a profile curve. Although our tool processes each frame independently, it achieves temporally continuous deformations with artistic control of the bulge through a small number of pseudo-stiffness parameters. The plausibility of the deformation is further enhanced by anisotropically spreading the volume-preserving bulge. An extension is also proposed to handle self-collisions between adjacent parts of the same object that often occur in character skinning animation.

The result of this work is a robust, real-time deformer that can handle complex geometric configurations like a ball squashed by a hand, colliding lips, bending fingers, etc.

Keywords: 3D animation, mesh deformation, collision response, computer graphics, geometry processing.

Unités de recherche
Inria Bordeaux - Sud-Ouest et UMR 5800 Université, 33000 Bordeaux, France.